Mathematics of Neural Networks

Volume VI: Reinforcement Learning and Sequential Decision Processes

Markov decision processes, policy optimisation, and the mathematics of learning from interaction

---

Every derivation step by step

Every symbol defined

Explicit numerical examples

Geometric intuitions

Silvano Salvador

Contents

89.1 The Reinforcement Learning Problem 6

89.2 Returns and Discount 7

89.3 Value Functions 9

89.4 The Bellman Equations 11

90.1 Optimal Value Functions and Policies 12

90.2 Policy Iteration 14

90.3 Value Iteration 15

91.1 Monte Carlo vs. Temporal Difference 17

91.2 TD(0): One-Step Prediction 19

91.3 SARSA and Q-Learning 20

91.4 Eligibility Traces and TD(lambda) 21

92.1 Function Approximation in RL 24

92.2 DQN: Experience Replay and Target Networks 25

92.3 Double DQN and Dueling Architecture 26

93.1 The Policy Gradient Theorem 27

93.2 REINFORCE and Variance Reduction 29

93.3 Actor-Critic Algorithms 30

94.1 Trust Region Policy Optimisation 31

94.2 Generalised Advantage Estimation 32

94.3 PPO: Clipped Surrogate Objective 34

95.1 The Maximum Entropy Framework 36

95.2 Soft Actor-Critic 37

95.3 Deterministic Policy Gradient 39

96.1 Model-Based vs. Model-Free RL 40

96.2 Dyna-Q: Integrating Learning and Planning 41

96.3 World Models and Latent Space Planning 43

97.1 Markov Games 44

97.2 Nash Equilibria in Markov Games 45
97.3 Cooperative MARL: CTDE 46
98.1 The Multi-Armed Bandit 47
98.2 UCB: Upper Confidence Bounds 48
98.3 Thompson Sampling 49
98.4 Exploration in MDPs 50
99.1 PAC-MDP Learning 51
99.2 Regret Bounds for Online RL 51
99.3 The Deadly Triad and Divergence 52
99.4 Connections to Supervised Learning 53
100.1 Distributional Reinforcement Learning 54
100.2 Offline Reinforcement Learning: Mathematics of Policy Pessimism 55
100.3 RLHF and Constitutional AI: Mathematics of Alignment 55
101.1 The Options Framework 56
101.2 Goal-Conditioned Policies and HER 57
101.3 Feudal Networks and Subgoal Discovery 58
102.1 The Meta-RL Problem 58
102.2 MAML for RL 59
102.3 RL2 and In-Context RL 60
103.1 Potential-Based Reward Shaping 61
103.2 Intrinsic Motivation 62
103.3 Curriculum Learning in RL 63
104.1 The Inverse Reinforcement Learning Problem 63
104.2 Maximum Entropy IRL 64
104.3 Behavioural Cloning vs. IRL 65
105.1 Constrained Markov Decision Processes 66
105.2 CPO and Safe Policy Improvement 68
105.3 Shield Synthesis and Formal Safety 68

106.1 The LLM-as-MDP Framework 69
106.2 Group Relative Policy Optimisation (GRPO) 70
106.3 Process Reward Models and Chain-of-Thought 71
106.4 Constitutional AI and RLAIF 72
107.1 Offline RL: The Distributional Shift Problem 73
107.2 The Pessimism Principle 74
107.3 IQL, TD3+BC, and Offline Actor-Critic 75
108.1 Continuous-Time MDPs 75
108.2 Hamilton-Jacobi-Bellman Equation 76
108.3 Pontryagin Maximum Principle 77
109.1 Information-Theoretic Lower Bounds 78
109.2 The Burn-in Period 79
109.3 Sample Efficiency in Deep RL 80
110.1 Bisimulation Metrics 81
110.2 Successor Representations 82
110.3 Contrastive Representation Learning 83
111.1 Wasserstein Distances and Distributional RL 84
111.2 KL-Regularised RL as Free Energy Minimisation 84
111.3 Value Functions as Log-Partition Functions 85
112.1 Sample Efficiency Gap 86
112.2 Reward Misspecification and Goodhart's Law 87
112.3 Causal Reinforcement Learning 88
112.4 Universal Reinforcement Learning and AI Safety 89
113.1 Complete Q-Learning Implementation 90
113.2 Policy Gradient on CartPole 91
113.3 PPO on MuJoCo 92
113.4 Deep Dive: Soft Actor-Critic Hyperparameters 93
113.5 Multi-Agent Example: Cooperative Navigation 93

113.6 Convergence Diagnostics in Practice 94

113.7 Reward Shaping Case Study: Sparse Fetch Environments 95

Notation Summary 98

Key Theorems Reference 100

Mathematical Background 102

Preface and Overview

This volume. Volume VI concludes the *Mathematics of Neural Networks* series with a rigorous treatment of reinforcement learning (RL) and sequential decision-making. Where earlier volumes covered feedforward and recurrent architectures, optimisation theory, graph and geometric deep learning, and advanced topics in scaling and alignment, this volume addresses the mathematics of learning from *interaction*: agents, environments, rewards, and the formal machinery of Markov decision processes.

Scope. The volume is organised into 25 Parts (89–113), covering four broad areas:

1. **Parts 89–92**: Foundations — MDPs, dynamic programming, temporal difference methods, and the transition to function approximation and deep RL.
2. **Parts 93–96**: Policy optimisation — policy gradient theorem, actor-critic methods, TRPO, PPO, maximum entropy RL, SAC, and model-based methods.
3. **Parts 97–99**: Multi-agent RL, exploration-exploitation, bandit algorithms, PAC-MDP learning theory, and the deadly triad.
4. **Parts 100–113**: Advanced topics — hierarchical RL, meta-RL, reward shaping, inverse RL, safe RL, offline RL, continuous-time control, causal RL, connections to free energy minimisation, and worked implementation examples.

Mathematical prerequisites. The reader is assumed comfortable with probability theory (measure-theoretic foundations helpful but not required), linear algebra, calculus, and the content of Volumes I–V. Key notation: $\mathbf{x}$ for vectors, $\mathbf{A}$ for matrices, $\mathbb{E}[\cdot]$ for expectation, ∇ for gradient, $\triangleq$ for definition.

Connection to previous volumes. The material here connects to:

- *Vol. I–II* (Feedforward networks): Q-networks, policy networks, and value function approximators are neural networks trained with the algorithms from those volumes.
- *Vol. III* (GNNs): Multi-agent RL over communication graphs uses GNN policy architectures.
- *Vol. IV* (Optimisation and theory): PAC-MDP bounds are analogous to PAC learning bounds; policy gradient methods are instances of stochastic optimisation.
- *Vol. V* (RLHF, alignment): Direct policy optimisation (DPO) and RLHF are introduced there as application-level algorithms; here we provide their rigorous RL foundations.

A note on notation. Throughout this volume, $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, π a policy, V^π the state-value function under π, Q^π the action-value function, A^π the advantage function, r a reward, and G a return (cumulative reward). The Bellman operators $\mathcal{T}^\pi$ and $\mathcal{T}^*$ appear throughout Parts 89–90.

PART 89

Foundations: Markov Decision Processes

We introduce the formal mathematical framework of Markov Decision Processes — the foundation of all reinforcement learning. Every element is defined precisely: state and action spaces, transition kernels, reward functions, and the central objects of value functions and optimal policies.

89.1 The Reinforcement Learning Problem

The agent-environment loop. Reinforcement learning formalises the problem of an *agent* choosing *actions* in an *environment* to maximise cumulative *reward*. At each discrete time step $t = 0, 1, 2, \ldots$:

$$s_t \in \mathcal{S} \xrightarrow{\pi} a_t \in \mathcal{A} \xrightarrow{\mathcal{P}} s_{t+1},\ r_t$$

The agent observes state s_t, selects action a_t according to policy π, receives scalar reward $r_t \in \mathbb{R}$, and transitions to s_{t+1}. The goal is to find π that maximises expected cumulative reward.

Markov Decision Process (MDP)

A *Markov Decision Process* is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where:

- $\mathcal{S}$ is the *state space* (finite, countable, or continuous);
- $\mathcal{A}$ is the *action space*;
- $\mathcal{P}(s' \mid s, a) = P(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the *transition kernel*;
- $\mathcal{R}(s, a) = \mathbb{E}[r_t \mid s_t = s, a_t = a]$ is the *reward function*;
- $\gamma \in [0, 1)$ is the *discount factor*.

The Markov property. The transition kernel $\mathcal{P}$ depends only on the current state and action, not on history:

$$P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = P(s_{t+1} \mid s_t, a_t)$$

This is the *Markov property*: the current state contains all information necessary to predict the future. It is the cornerstone assumption that makes MDPs tractable.

Policies. A *policy* π specifies how the agent acts. The two main classes are:

- **Deterministic policy**: $\pi : \mathcal{S} \to \mathcal{A}$, a function $a = \pi(s)$.
- **Stochastic policy**: $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, a conditional distribution $\pi(a \mid s) = P(a_t = a \mid s_t = s)$.

Stochastic policies are strictly more general and are essential for environments with partial observability or for policy gradient methods.

Partial observability and POMDPs. In many real-world settings the agent does not observe the full state s_t but only a noisy signal $o_t \in \mathcal{O}$. This leads to the *Partially Observable Markov Decision Process* (POMDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{O}_{\text{fn}}, \gamma)$, where the *observation function* is:

$$\mathcal{O}_{\text{fn}}(o \mid s', a) = P(o_t = o \mid s_t = s', a_{t-1} = a)$$

The agent cannot act on s_t directly; instead it maintains a *belief state* $b_t(s) = P(s_t = s \mid o_{1:t}, a_{0:t-1})$, which is a probability distribution over $\mathcal{S}$. The belief updates via Bayes' rule:

$$b_{t+1}(s') = \frac{\mathcal{O}_{\text{fn}}(o_{t+1} \mid s', a_t) \sum_s \mathcal{P}(s' \mid s, a_t)\, b_t(s)}{\sum_{\hat{s}'} \mathcal{O}_{\text{fn}}(o_{t+1} \mid \hat{s}', a_t) \sum_s \mathcal{P}(\hat{s}' \mid s, a_t)\, b_t(s)}$$

The space of beliefs $\mathcal{B} = \{\text{prob. distributions over } \mathcal{S}\}$ is a continuous simplex; a POMDP reduces to an MDP over $\mathcal{B}$ (the *belief MDP*), but this state space is infinite-dimensional and generally intractable to solve exactly.

Reward Hypothesis

All goals and purposes of the agent can be well described by the maximisation of the expected value of the cumulative sum of a received scalar signal (reward). This hypothesis (Sutton & Barto, 2018) is the foundational claim of RL. Its correctness is not provable in general, but it is a remarkably powerful simplification: with the right reward function, complex behaviours (locomotion, language generation, game playing) emerge from this single objective.

Continuous vs. discrete spaces.

- **Discrete** $\mathcal{S}$, $\mathcal{A}$: finite MDPs, tabular algorithms apply. $|\mathcal{S}| \cdot |\mathcal{A}|$ state-action pairs; exact value functions representable as tables.
- **Continuous** $\mathcal{S} \subseteq \mathbb{R}^n$: requires function approximation. Transition kernel becomes a conditional density $\mathcal{P}(s' \mid s, a)$.
- **Continuous** $\mathcal{A} \subseteq \mathbb{R}^m$: cannot enumerate actions for $\max_a$; policy gradient or actor-critic methods required.

Example — 2-State MDP: complete specification

Consider $\mathcal{S} = \{s_1, s_2\}$, $\mathcal{A} = \{a_L, a_R\}$, $\gamma = 0.9$.

Transition matrices (rows = current state, columns = next state):

$$P(a_L) = \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{pmatrix}, \qquad P(a_R) = \begin{pmatrix} 0.2 & 0.8 \\ 0.1 & 0.9 \end{pmatrix}$$

Reward function: $\mathcal{R}(s_1, a_L) = 5$, $\mathcal{R}(s_1, a_R) = 1$, $\mathcal{R}(s_2, a_L) = 0$, $\mathcal{R}(s_2, a_R) = 10$.

Deterministic policy π_1: always take a_L. Policy-weighted transition matrix: $P^{\pi_1} = P(a_L)$. Reward vector: $R^{\pi_1} = [5, 0]^\top$. System $(I - 0.9P^{\pi_1})v = R^{\pi_1}$:

$$\begin{pmatrix} 1-0.63 & -0.27 \\ -0.36 & 1-0.54 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0.37 & -0.27 \\ -0.36 & 0.46 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \end{pmatrix}$$

$\det = 0.37 \times 0.46 - 0.27 \times 0.36 = 0.1702 - 0.0972 = 0.0730$. $v_1 = (5 \times 0.46)/0.073 \approx 31.5$; $v_2 = (5 \times 0.36)/0.073 \approx 24.7$.

Stochastic policies and their necessity. While deterministic policies suffice for MDPs (proved in §90.1), stochastic policies $\pi(a \mid s)$ are: (a) essential for POMDPs (the optimal policy over belief states may require mixed strategies); (b) necessary for policy gradient methods (gradients require $\log \pi$ to be differentiable); (c) preferred in multi-agent settings (Nash equilibria are generically mixed).

The set of stochastic policies is a convex set (convex combination of two policies is a valid policy), whereas deterministic policies form the extreme points of this convex set.

89.2 Returns and Discount

The return. The *return* G_t is the total discounted future reward from time t:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

Why discount? Discounting $\gamma < 1$ serves three purposes: (1) it ensures G_t is finite for bounded rewards ($|G_t| \leq r_{\max}/(1-\gamma)$); (2) it models the economic notion that future rewards are less certain; (3) it creates a preference for sooner rewards, which aids learning convergence.

Example — Computing returns with $\gamma = 0.9$

Suppose the agent receives rewards $r_0 = 1,\ r_1 = 2,\ r_2 = 3$ and then terminates. The return from $t = 0$ is:

$$G_0 = 1 + 0.9 \cdot 2 + 0.9^2 \cdot 3 = 1 + 1.8 + 2.43 = 5.23$$

From $t = 1$: $G_1 = 2 + 0.9 \cdot 3 = 4.7$. Note the recursive relation $G_t = r_t + \gamma G_{t+1}$, which is the basis of all dynamic programming algorithms.

Episodic vs. continuing tasks.

- **Episodic**: interaction splits into episodes of finite length T, ending in a terminal state. Return is $G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k}$.
- **Continuing**: no terminal state; $\gamma < 1$ ensures finite return.

Boundedness of returns. For bounded rewards $|r_t| \leq r_{\max}$ and discount $\gamma < 1$:

$$|G_t| = \left| \sum_{k=0}^{\infty} \gamma^k r_{t+k} \right| \leq \sum_{k=0}^{\infty} \gamma^k |r_{t+k}| \leq r_{\max} \sum_{k=0}^{\infty} \gamma^k = \frac{r_{\max}}{1-\gamma}$$

This geometric series convergence is the fundamental reason discounting makes the RL problem well-posed.

The recursive structure. The return satisfies:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \cdots) = r_t + \gamma G_{t+1}$$

This *Bellman recursion* for the return is the foundation of all dynamic programming in RL. It means: to estimate G_t, we only need r_t and an estimate of G_{t+1}.

Average reward formulation. For continuing tasks where $\gamma \to 1$, the discounted return diverges. Instead, use the *average reward*:

$$\rho(\pi) \triangleq \lim_{T \to \infty} \frac{1}{T} \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} r_t \right] = \sum_s d^\pi(s) \sum_a \pi(a \mid s)\, \mathcal{R}(s,a)$$

where $d^\pi(s)$ is the stationary distribution of the Markov chain induced by π (assuming ergodicity). The *differential value function* $\tilde{V}^\pi : \mathcal{S} \to \mathbb{R}$ satisfies the *average reward Bellman equation*:

$$\tilde{V}^\pi(s) = \sum_a \pi(a \mid s) \left[\mathcal{R}(s,a) - \rho(\pi) + \sum_{s'} \mathcal{P}(s' \mid s, a)\, \tilde{V}^\pi(s') \right]$$

Unlike the discounted case, $\tilde{V}^\pi$ is defined only up to a constant (adding any constant c to all values gives another solution), requiring a reference state s_0 with $\tilde{V}^\pi(s_0) = 0$.

Example — Sensitivity of return to γ

Rewards: $r_0 = 10,\ r_1 = 0,\ r_2 = 0,\ r_3 = 10,\ r_4 = 0,\ r_5 = 10, \ldots$ (reward every 3 steps).

With $\gamma = 0.5$: $G_0 = 10 + 0 + 0 + 0.125 \cdot 10 + 0 + 0.03125 \cdot 10 + \cdots = 10 \cdot (1 + 0.5^3 + 0.5^6 + \cdots) = 10/(1 - 0.125) \approx 11.43$.
With $\gamma = 0.99$: $G_0 = 10 \cdot (1 + 0.99^3 + 0.99^6 + \cdots) = 10/(1 - 0.97030) \approx 10/0.02970 \approx 336.7$.
The agent with $\gamma = 0.99$ values the long-run reward stream *29 times more* than with $\gamma = 0.5$. The choice of γ has a dramatic effect on policy and behaviour: high γ creates far-sighted agents that plan over long horizons; low γ creates myopic agents that prioritise immediate rewards.

Episodic return in matrix form. For an episode of length T, stack rewards $\mathbf{r} = [r_0, \ldots, r_{T-1}]^\top$. The return vector $\mathbf{G} = [G_0, \ldots, G_{T-1}]^\top$ satisfies $\mathbf{G} = \mathbf{D}_\gamma \mathbf{r}$ where $\mathbf{D}_\gamma$ is the upper-triangular matrix with $(\mathbf{D}_\gamma)_{tk} = \gamma^{k-t}$ for $k \geq t$. This is the *return matrix*, useful for vectorised implementations of REINFORCE.

89.3 Value Functions

The state-value function. Under policy π, the *state-value function* $V^\pi : \mathcal{S} \to \mathbb{R}$ is the expected return starting from state s:

$$V^\pi(s) \triangleq \mathbb{E}_\pi[G_t \mid s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, s_t = s\right]$$

The action-value function. The *action-value function* (or *Q-function*) $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the expected return starting from s, taking action a, then following π:

$$Q^\pi(s, a) \triangleq \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a]$$

Relation between V and Q.

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, Q^\pi(s, a)$$

and conversely:

$$Q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a)\, V^\pi(s')$$

The advantage function. The *advantage function* $A^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ measures how much better action a is relative to the average:

$$A^\pi(s, a) \triangleq Q^\pi(s, a) - V^\pi(s)$$

By construction, $\mathbb{E}_{a \sim \pi}[A^\pi(s, a)] = 0$ for all s.

Existence and uniqueness of V^π via Banach's theorem. Define the *Bellman expectation operator* $\mathcal{T}^\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$:

$$(\mathcal{T}^\pi v)(s) \triangleq \sum_a \pi(a \mid s) \left[\mathcal{R}(s, a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s, a)\, v(s')\right]$$

Theorem: $\mathcal{T}^\pi$ is a γ-contraction

For any $u, v \in \mathbb{R}^{|\mathcal{S}|}$:

$$\left\|\mathcal{T}^\pi u - \mathcal{T}^\pi v\right\|_\infty \leq \gamma \left\|u - v\right\|_\infty$$

Proof.

$$|(\mathcal{T}^\pi u)(s) - (\mathcal{T}^\pi v)(s)| = \gamma \left|\sum_a \pi(a|s) \sum_{s'} \mathcal{P}(s'|s,a)(u(s') - v(s'))\right| \leq \gamma \left\|u - v\right\|_\infty$$

since $\sum_a \pi(a|s) \sum_{s'} \mathcal{P}(s'|s,a) = 1$. By Banach's fixed-point theorem, $\mathcal{T}^\pi$ has a unique fixed point $V^\pi = \mathcal{T}^\pi V^\pi$, and the iterates $v_{k+1} = \mathcal{T}^\pi v_k$ converge: $\left\|v_k - V^\pi\right\|_\infty \leq \gamma^k \left\|v_0 - V^\pi\right\|_\infty$. □

Matrix form and explicit solution. Let $\mathbf{P}^\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ with $P^\pi_{ss'} = \sum_a \pi(a|s)\mathcal{P}(s'|s,a)$, and $\mathbf{R}^\pi_s = \sum_a \pi(a|s)\mathcal{R}(s,a)$. Then:

$$\mathbf{v}^\pi = \mathbf{R}^\pi + \gamma \mathbf{P}^\pi \mathbf{v}^\pi \implies (\mathbf{I} - \gamma \mathbf{P}^\pi)\mathbf{v}^\pi = \mathbf{R}^\pi \implies \mathbf{v}^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1}\mathbf{R}^\pi$$

The matrix $(\mathbf{I} - \gamma \mathbf{P}^\pi)$ is invertible because $\mathbf{P}^\pi$ is a stochastic matrix (all eigenvalues of $\mathbf{P}^\pi$ have modulus ≤ 1, so all eigenvalues of $\gamma \mathbf{P}^\pi$ lie strictly inside the unit disk), giving $\det(\mathbf{I} - \gamma \mathbf{P}^\pi) \neq 0$. Moreover:

$$(\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} = \sum_{k=0}^{\infty} (\gamma \mathbf{P}^\pi)^k = \mathbf{I} + \gamma \mathbf{P}^\pi + \gamma^2 (\mathbf{P}^\pi)^2 + \cdots$$

The geometric series interpretation: $[(\gamma \mathbf{P}^\pi)^k]_{ss'}$ is γ^k times the probability of being in state s' after k steps under π starting from s. Therefore:

$$V^\pi(s) = \sum_{k=0}^{\infty} \gamma^k \sum_{s'} P^{\pi,k}_{ss'} \sum_a \pi(a|s')\mathcal{R}(s',a)$$

where $P^{\pi,k}$ is the k-step transition matrix. This is the *occupancy measure* representation.

Occupancy measure. Define $d^\pi_s(s') \triangleq (1-\gamma)\sum_{k=0}^{\infty} \gamma^k P^{\pi,k}_{ss'}$, the discounted frequency of visiting s' starting from s. Then:

$$V^\pi(s) = \frac{1}{1-\gamma} \mathbb{E}_{s' \sim d^\pi_s}\left[\sum_a \pi(a|s')\mathcal{R}(s',a)\right]$$

Example — 3-state MDP: computing V^π and Q^π by matrix inversion

$\mathcal{S} = \{1,2,3\}$, deterministic π: state $1 \to a_1$, state $2 \to a_2$, state $3 \to a_1$. $\gamma = 0.8$. Rewards: $\mathcal{R}(1,a_1) = 2$, $\mathcal{R}(2,a_2) = 0$, $\mathcal{R}(3,a_1) = 4$.

$$\mathbf{P}^\pi = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{R}^\pi = \begin{pmatrix} 2 \\ 0 \\ 4 \end{pmatrix}$$

$$\mathbf{I} - 0.8\mathbf{P}^\pi = \begin{pmatrix} 1 & -0.4 & -0.4 \\ -0.8 & 1 & 0 \\ 0 & 0 & 0.2 \end{pmatrix}$$

From row 3: $v_3 = 4/0.2 = 20$. From row 2: $v_2 = 0.8v_1$. Row 1: $v_1 - 0.4(0.8v_1) - 0.4(20) = 2$, so $v_1(1 - 0.32) = 2 + 8 = 10$, giving $v_1 = 10/0.68 \approx 14.7$, $v_2 \approx 11.8$, $v_3 = 20$.

89.4 The Bellman Equations

The Bellman equations are the central recursive characterisation of value functions and the algebraic foundation of most RL algorithms.

Bellman Expectation Equation

For any policy π:

$$V^\pi(s) = \sum_a \pi(a \mid s) \left[\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a)\, V^\pi(s') \right]$$

$$Q^\pi(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, Q^\pi(s',a')$$

These equations can be written in matrix-vector form. Let $\mathbf{v}^\pi \in \mathbb{R}^{|\mathcal{S}|}$ be the vector of state values, $\mathbf{R}^\pi \in \mathbb{R}^{|\mathcal{S}|}$ the expected reward vector, and $\mathbf{P}^\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ the policy-weighted transition matrix with entries $P^\pi_{ss'} = \sum_a \pi(a \mid s)\mathcal{P}(s' \mid s,a)$. Then:

$$\mathbf{v}^\pi = \mathbf{R}^\pi + \gamma \mathbf{P}^\pi \mathbf{v}^\pi \implies \mathbf{v}^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} \mathbf{R}^\pi$$

The matrix $(\mathbf{I} - \gamma \mathbf{P}^\pi)$ is invertible because $\gamma < 1$ and $\mathbf{P}^\pi$ is a stochastic matrix with spectral radius 1, so all eigenvalues of $\gamma \mathbf{P}^\pi$ lie strictly inside the unit disk.

Example — Bellman equation for a 3-state MDP

Consider $\mathcal{S} = \{s_1, s_2, s_3\}$, deterministic policy π giving a_1 always, $\gamma = 0.9$, rewards $\mathcal{R} = [1, 0, 5]^\top$, and transition matrix:

$$\mathbf{P}^\pi = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Then $\mathbf{v}^\pi = (\mathbf{I} - 0.9\,\mathbf{P}^\pi)^{-1}[1,0,5]^\top$. The system $\mathbf{v} = \mathbf{R} + 0.9\mathbf{P}\mathbf{v}$ gives:

$$v_1 = 1 + 0.9 v_2, \quad v_2 = 0 + 0.9 v_3, \quad v_3 = 5 + 0.9 v_1$$

Substituting: $v_1 = 1 + 0.9(0.9(5 + 0.9 v_1)) = 1 + 4.05 + 0.729 v_1$, giving $v_1 = 5.05 / 0.271 \approx 18.6$. Then $v_3 \approx 21.7$, $v_2 \approx 19.6$.

Derivation from first principles. By the law of total expectation, conditioning on the first action and next state:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[G_t \mid s_t = s] = \mathbb{E}_\pi[r_t + \gamma G_{t+1} \mid s_t = s] \\ &= \sum_a \pi(a \mid s)\, \mathbb{E}_\pi[r_t + \gamma G_{t+1} \mid s_t = s, a_t = a] \\ &= \sum_a \pi(a \mid s) \left[\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a)\, \mathbb{E}_\pi[G_{t+1} \mid s_{t+1} = s'] \right] \\ &= \sum_a \pi(a \mid s) \left[\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a)\, V^\pi(s') \right] \end{aligned}$$

The last step uses the Markov property: $\mathbb{E}_\pi[G_{t+1} \mid s_{t+1} = s'] = V^\pi(s')$ regardless of history.

One-step backup diagram. The Bellman equation encodes a *backup*: the value of a state is computed by looking ahead one step, averaging over actions (policy), then over next states (transitions), and bootstrapping from the next-state values.

Policy evaluation as fixed-point iteration. Initialise $v_0 = \mathbf{0}$. Iterate:

$$v_{k+1}(s) = \sum_a \pi(a|s) \left[\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|s,a)\, v_k(s') \right]$$

Since $\mathcal{T}^\pi$ is a γ-contraction: $\|v_{k+1} - V^\pi\|_\infty = \|\mathcal{T}^\pi v_k - \mathcal{T}^\pi V^\pi\|_\infty \leq \gamma \|v_k - V^\pi\|_\infty$, so convergence is linear with rate γ.

Example — 4-state gridworld: 3 iterations of policy evaluation

Grid: states $\{1,2,3,4\}$ in a line (1-2-3-4). Terminal state 4 with $V(4) = 0$ always. Policy: always move right. $\mathcal{R}(s, \text{right}) = -1$ for $s = 1, 2, 3$. $\gamma = 1.0$ (undiscounted episodic).

Iteration 0: $v_0 = [0, 0, 0, 0]$.
Iteration 1: $v_1(3) = -1 + 1.0 \cdot v_0(4) = -1$; $v_1(2) = -1 + v_0(3) = -1$; $v_1(1) = -1 + v_0(2) = -1$.
Iteration 2: $v_2(3) = -1 + v_1(4) = -1$; $v_2(2) = -1 + v_1(3) = -2$; $v_2(1) = -1 + v_1(2) = -2$.
Iteration 3: $v_3 = [-3, -2, -1, 0]$. This has converged to the true $V^\pi = [-3, -2, -1, 0]$ (exactly k steps to terminal from state $k - 3$).

Connection to dynamic programming. Bellman's *principle of optimality* states: an optimal policy has the property that, whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. This is precisely the recursive structure of the Bellman equation, and it is the reason DP and RL can solve sequential decision problems efficiently: a problem of horizon T is decomposed into T subproblems of horizon 1.

Part 90

Optimal Control: Bellman Optimality

We derive the Bellman optimality equations and the conditions under which optimal policies exist and are unique. Dynamic programming algorithms — policy iteration and value iteration — are presented with full convergence proofs.

90.1 Optimal Value Functions and Policies

Partial ordering on policies. Policy π' dominates π if $V^{\pi'}(s) \geq V^\pi(s)$ for all $s \in \mathcal{S}$. The *optimal policy* π^* satisfies $V^{\pi^*}(s) \geq V^\pi(s)$ for all s and all π.

Theorem: Existence of optimal policy

For any finite MDP, there exists at least one deterministic optimal policy π^*, and all optimal policies share the same optimal value function V^* and optimal action-value function Q^*.

Bellman Optimality Equations

$$V^*(s) = \max_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a)\, V^*(s') \right]$$

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a)\, \max_{a'} Q^*(s',a')$$

The optimal policy is obtained greedily:

$$\pi^*(s) = \arg\max_a Q^*(s,a) = \arg\max_a \left[\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a)\, V^*(s') \right]$$

The Bellman optimality operator. Define the operator $\mathcal{T} : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$:

$$(\mathcal{T}v)(s) \triangleq \max_a \left[\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a)\, v(s') \right]$$

Theorem: Contraction

$\mathcal{T}$ is a γ-contraction in the ℓ^∞ norm:

$$\|\mathcal{T}u - \mathcal{T}v\|_\infty \le \gamma \|u - v\|_\infty \qquad \forall u, v \in \mathbb{R}^{|\mathcal{S}|}$$

By Banach's fixed-point theorem, $\mathcal{T}$ has a unique fixed point $V^* = \mathcal{T}V^*$, and iterating $v_{k+1} = \mathcal{T}v_k$ converges to V^* for any initialisation v_0.

Proof. For any u, v and state s:

$$|(\mathcal{T}u)(s) - (\mathcal{T}v)(s)| = \left| \max_a [\dots + \gamma \sum_{s'} \mathcal{P}(s'|s,a) u(s')] - \max_a [\dots + \gamma \sum_{s'} \mathcal{P}(s'|s,a) v(s')] \right|$$

Using $|\max_a f(a) - \max_a g(a)| \le \max_a |f(a) - g(a)|$ and the fact that $\mathcal{P}$ is a probability distribution:

$$\le \gamma \max_a \sum_{s'} \mathcal{P}(s' \mid s,a) |u(s') - v(s')| \le \gamma \|u - v\|_\infty$$

Taking supremum over s gives $\|\mathcal{T}u - \mathcal{T}v\|_\infty \le \gamma \|u - v\|_\infty$. □

Well-definedness of V^*. The partial order $\pi' \ge \pi$ (meaning $V^{\pi'}(s) \ge V^\pi(s)\ \forall s$) is well-founded on the finite set of policies. To prove $V^* = \max_\pi V^\pi$ exists, note that the set of stationary deterministic policies is finite (cardinality $|\mathcal{A}|^{|\mathcal{S}|}$), so the maximum is attained.

Principle of Optimality (Bellman, 1957)

A policy π^* is optimal if and only if, for every state $s \in \mathcal{S}$:

$$V^*(s) = \max_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a)\, V^*(s') \right]$$

Proof sketch. ($\Rightarrow$) If π^* is optimal, then for all s: $V^*(s) = V^{\pi^*}(s) = \max_a [\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|s,a) V^*(s')]$, otherwise we could improve π^* by changing the action at s, contradicting optimality. ($\Leftarrow$)

If v satisfies the Bellman optimality equation, then $v = V^*$ (unique fixed point of $\mathcal{T}$, which is a γ-contraction by the contraction theorem in §90.1).

Why deterministic policies suffice. For a stochastic policy $\pi(a \mid s)$:

$$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s,a) \le \max_a Q^\pi(s,a) = V^{\pi'}(s)$$

where π' is the deterministic policy selecting $\arg\max_a Q^\pi(s,a)$. So any stochastic policy is dominated by some deterministic one. This means that in the search for π^*, we need only consider $|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies.

Example — 4-state MDP: Bellman optimality equations by hand

$\mathcal{S} = \{1,2,3,4\}$, $\mathcal{A} = \{L, R\}$, $\gamma = 0.9$. State 4 is terminal ($V^*(4) = 0$). Transitions: from state s, L goes to $\max(s-1, 1)$ deterministically; R goes to $s+1$ (if $s < 4$) or stays. Rewards: $r(s,R) = 0$, $r(3,R) = +10$ (reaching terminal gives reward when transitioning to 4).

Iteration 0: $v_0 = [0,0,0,0]$.
Iteration 1 (Bellman optimality): $v_1(3) = \max[r(3,L)+0.9v_0(2),\ r(3,R)+0.9v_0(4)] = \max[0, 10] = 10$. $v_1(2) = \max[0 + 0.9 \cdot 0, 0 + 0.9 \cdot 10] = \max[0, 9] = 9$. $v_1(1) = \max[0, 0 + 0.9 \cdot 9] = 8.1$.
Iteration 2: $v_2(3) = \max[0.9v_1(2), 10] = \max[8.1, 10] = 10$. $v_2(2) = \max[0.9v_1(1), 0.9v_1(3)] = \max[7.29, 9] = 9$. $v_2(1) = \max[0.9 \cdot 0, 0.9 \cdot 9] = 8.1$.
Converging to $V^* = [8.1, 9, 10, 0]$ with optimal policy: always go R.

90.2 Policy Iteration

Policy iteration alternates between two steps until convergence:

1. **Policy evaluation**: given π_k, solve $\mathbf{v}^{\pi_k} = (\mathbf{I} - \gamma \mathbf{P}^{\pi_k})^{-1} \mathbf{R}^{\pi_k}$ (exactly, or via iterative methods).
2. **Policy improvement**: set $\pi_{k+1}(s) = \arg\max_a \left[\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|s,a)\, V^{\pi_k}(s')\right]$.

Policy Improvement Theorem

If π_{k+1} is the greedy improvement of π_k, then $V^{\pi_{k+1}}(s) \ge V^{\pi_k}(s)$ for all s, with equality if and only if $\pi_k = \pi^*$.

Proof. By the greedy construction, $Q^{\pi_k}(s, \pi_{k+1}(s)) \ge V^{\pi_k}(s)$ for all s. Expanding:

$$V^{\pi_k}(s) \le Q^{\pi_k}(s, \pi_{k+1}(s)) = \mathcal{R}(s, \pi_{k+1}(s)) + \gamma \sum_{s'} \mathcal{P}(s'|s, \pi_{k+1}(s))\, V^{\pi_k}(s')$$

Applying the same inequality recursively:

$$\le \mathcal{R}(s, \pi_{k+1}(s)) + \gamma \sum_{s'} \mathcal{P}(s'|s, \pi_{k+1}(s)) \left[\mathcal{R}(s', \pi_{k+1}(s')) + \gamma \sum_{s''} \cdots\right] = V^{\pi_{k+1}}(s)$$

The sequence $\{V^{\pi_k}\}$ is monotone non-decreasing. Since the number of deterministic policies is finite ($|\mathcal{A}|^{|\mathcal{S}|}$), policy iteration terminates in a finite number of steps. □

Full policy iteration pseudocode.

Policy Iteration

Initialise π_0 arbitrarily. Repeat until convergence:

1. **Policy evaluation**: solve $(I-\gamma P^{\pi_k})v = R^{\pi_k}$ for $v = V^{\pi_k}$ (exact: matrix inversion; approximate: iterate $v_{j+1} = R^{\pi_k} + \gamma P^{\pi_k} v_j$ until $\|v_{j+1} - v_j\|_\infty < \theta$).
2. **Policy improvement**: $\pi_{k+1}(s) = \arg\max_a [\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|s,a)\, V^{\pi_k}(s')]$.
3. If $\pi_{k+1} = \pi_k$: return $\pi_k = \pi^*$.

Modified policy iteration (truncated evaluation). Instead of solving exactly, do only m sweeps of the Bellman operator in step 1. For $m = 1$, this reduces to value iteration; as $m \to \infty$, it approaches exact policy iteration. For practical implementations, $m \in \{5, 10, 20\}$ often balances accuracy and computation.

Computational complexity.

- **Exact policy evaluation**: $O(|\mathcal{S}|^3)$ for matrix inversion (or $O(|\mathcal{S}|^2)$ per sweep of iterative evaluation).
- **Policy improvement**: $O(|\mathcal{S}| \cdot |\mathcal{A}| \cdot |\mathcal{S}|)$ per step (compute $Q(s,a)$ for all (s,a) pairs and take arg max).
- **Total**: $O(|\mathcal{S}|^2|\mathcal{A}|)$ per iteration; $O(|\mathcal{A}|^{|\mathcal{S}|})$ iterations in the worst case, but in practice $O(|\mathcal{S}|/(1-\gamma))$ iterations suffice.

Finite termination proof (detailed). The sequence $\pi_0, \pi_1, \pi_2, \ldots$ is strictly improving: $V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s)$ for all s (Policy Improvement Theorem), with strict inequality for at least one s unless $\pi_k = \pi^*$. Since the number of deterministic policies is finite ($|\mathcal{A}|^{|\mathcal{S}|}$), and each π_k is different from π_{k-1} (else we would have stopped), the sequence must terminate. Therefore policy iteration terminates in at most $|\mathcal{A}|^{|\mathcal{S}|}$ iterations — finite. □

Example — Policy iteration on a 3-state loop

$\mathcal{S} = \{A, B, C\}$, cycle: $A \to B \to C \to A$. Actions $\{0, 1\}$: action 0 moves forward, action 1 stays. $r(A,0) = 0$, $r(B,0) = 2$, $r(C,0) = 1$, all stay-rewards $= 0$. $\gamma = 0.9$.
π_0: all-stay (action 1). $V^{\pi_0} = [0, 0, 0]$ (no reward ever).
Improvement: $Q(A,0) = 0 + 0.9V(B) = 0 > Q(A,1) = 0$. Both equal; break tie: choose 0. $Q(B,0) = 2 + 0.9V(C) = 2 > 0$. $Q(C,0) = 1 + 0.9V(A) = 1 > 0$. So π_1: all-forward.
Evaluate π_1: $V = R + 0.9P^{\pi_1}V$ with $P^{\pi_1}_{AB} = P^{\pi_1}_{BC} = P^{\pi_1}_{CA} = 1$. By symmetry of the cycle: $V(A) = 0 + 0.9V(B)$, $V(B) = 2 + 0.9V(C)$, $V(C) = 1 + 0.9V(A)$. Solving: $V(A) = 0.9(2 + 0.9(1 + 0.9V(A))) = 1.8 + 0.81 + 0.729V(A)$, so $V(A) = 2.61/0.271 \approx 9.63$. $V(B) \approx 10.7$, $V(C) \approx 9.67$.
Improvement: stays all-forward. Policy iteration converged in 1 step.

90.3 Value Iteration

Value iteration directly applies the Bellman optimality operator repeatedly:

$$v_{k+1}(s) = \max_a \left[\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|s,a)\, v_k(s') \right]$$

By the contraction theorem, $\|v_k - V^*\|_\infty \leq \gamma^k \|v_0 - V^*\|_\infty \to 0$.

Stopping criterion. In practice, stop when $\|v_{k+1} - v_k\|_\infty < \varepsilon(1-\gamma)/(2\gamma)$, which guarantees $\|v_k - V^*\|_\infty < \varepsilon/2$.

Example — Value iteration on a grid world

A 3×3 grid, goal state at $(3,3)$ with $r = +1$, all other transitions $r = 0$, walls absorbing. With $\gamma = 0.95$ and $v_0 = \mathbf{0}$:

After iteration 1: $v(2,3) = v(3,2) = 0.95$, all others 0.
After iteration 2: $v(1,3) = v(2,2) = v(3,1) = 0.95^2 = 0.9025$, $v(2,3) = v(3,2) = 0.95 + 0.95^2 \cdot 0 = 0.95$.
Convergence is reached when values stabilise across all states.

Value iteration update rule. Starting from any v_0, apply:

$$v_{k+1}(s) = (\mathcal{T} v_k)(s) = \max_a \left[\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|s,a)\, v_k(s') \right]$$

By the contraction: $\|v_k - V^*\|_\infty \leq \gamma^k \|v_0 - V^*\|_\infty$. To achieve $\|v_k - V^*\|_\infty \leq \varepsilon$, it suffices to run $k \geq \frac{\log(\|v_0 - V^*\|_\infty / \varepsilon)}{\log(1/\gamma)}$ iterations.

Stopping criterion derivation. We want to stop when v_k is close enough to V^*. We can bound $\|v_k - V^*\|_\infty$ using only $\|v_{k+1} - v_k\|_\infty$:

$$\|v_k - V^*\|_\infty \leq \|v_k - v_{k+1}\|_\infty + \|v_{k+1} - V^*\|_\infty \leq \|v_k - v_{k+1}\|_\infty + \gamma \|v_k - V^*\|_\infty$$

Rearranging: $\|v_k - V^*\|_\infty \leq \frac{\|v_k - v_{k+1}\|_\infty}{1-\gamma}$. To ensure $\|v_k - V^*\|_\infty \leq \varepsilon$, stop when:

$$\|v_{k+1} - v_k\|_\infty \leq \varepsilon(1-\gamma)$$

Asynchronous value iteration. Instead of updating all states simultaneously, update one state at a time (in any order). Convergence is still guaranteed as long as every state is updated infinitely often. This is crucial for large state spaces: focus computation on relevant states.

Prioritised sweeping. Order updates by the *Bellman error*: $\delta(s) = \max_a [\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|s,a) v(s')] - v(s)$. Update states with largest $|\delta(s)|$ first. This directs computation where value information propagates most rapidly, achieving much faster convergence in practice.

Example — 3×3 grid world: 5 iterations of value iteration

Grid: states (i,j), $i,j \in \{1,2,3\}$. Actions: $\{N, S, E, W\}$, deterministic, walls reflect. Goal: state $(3,3)$, $r = +1$ on entering, $\gamma = 0.95$. Start: $v_0 = \mathbf{0}$.

Iter 1: Only $(2,3)$ and $(3,2)$ can reach $(3,3)$ in one step. $v_1(2,3) = 0 + 0.95 \times 1 = 0.95$; $v_1(3,2) = 0.95$.
Iter 2: $(1,3), (2,2), (3,1)$ now have $v_2 = 0.95 \times 0.95 = 0.9025$. $(2,3), (3,2)$: $v_2 = 0.95 \times 1 = 0.95$ (direct step still best).
Iter 3: Propagating further. $v_3(1,2) = 0.95^3 = 0.857$; $v_3(1,3) = 0.95 \times 0.9025 = 0.857$; diagonal states converge at same rate.
Iter 5: Values form a gradient increasing toward $(3,3)$. State $(1,1)$ has $v_5 = 0.95^4 \approx 0.815$. Converged values (geometric series): $V^*(i,j) = 0.95^{d(i,j)}$ where d is Manhattan distance to goal.

	$j=1$	$j=2$	$j=3$
$i=3$	0.857	0.950	1.000
$i=2$	0.815	0.902	0.950
$i=1$	0.774	0.815	0.857

PART 91

Temporal Difference Learning

Temporal difference (TD) methods combine ideas from Monte Carlo sampling and dynamic programming. They learn value functions from experience without a model of the environment, and form the backbone of practical RL algorithms.

91.1 Monte Carlo vs. Temporal Difference

The fundamental challenge. In practice, $\mathcal{P}$ and $\mathcal{R}$ are unknown. The agent must estimate value functions from sampled trajectories $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$.

Monte Carlo (MC) estimation. After completing an episode, estimate $V^\pi(s)$ as the sample mean of observed returns from visits to s:

$$V(s) \leftarrow V(s) + \alpha\left[G_t - V(s)\right]$$

where $G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k}$ is the observed return. MC is unbiased but high variance (returns depend on many future random actions).

The TD idea. Rather than waiting for the episode to end, *bootstrap*: use the current estimate $V(s_{t+1})$ as a proxy for future returns. The *TD target* is:

$$\hat{y}_t \triangleq r_t + \gamma V(s_{t+1})$$

The *TD error* δ_t is:

$$\delta_t \triangleq r_t + \gamma V(s_{t+1}) - V(s_t)$$

First-visit vs. every-visit Monte Carlo. For a state s, within a single episode, two variants estimate $V^\pi(s)$:

- **First-visit MC**: use the return G_t from the *first* visit to s in the episode.
- **Every-visit MC**: use the return from *every* visit to s.

Both are unbiased ($\mathbb{E}[G_t | s_t = s] = V^\pi(s)$) and converge to $V^\pi(s)$ by the law of large numbers. First-visit MC produces independent samples (each episode contributes at most one estimate per state); every-visit MC uses correlated samples (the returns G_t and $G_{t'}$ for two visits $t < t'$ to s share all rewards after t').

Incremental update derivation. Suppose $V_n(s)$ is the estimate after n visits. The update from a new return G_n:

$$V_{n+1}(s) = V_n(s) + \frac{1}{n+1}(G_n - V_n(s))$$

This is equivalent to the running mean. Replacing $1/(n+1)$ with a constant α gives the TD-style update: $V(s) \leftarrow V(s) + \alpha(G - V(s))$. This is exactly the MC update rule — TD(0) has the same form but uses a one-step bootstrap target.

Off-policy MC with importance sampling. Suppose we collect data under a behaviour policy μ but want to evaluate target policy π. The *importance sampling ratio* for a trajectory $\tau = (s_0, a_0, \dots, s_{T-1})$:

$$\rho_{0:T-1} = \prod_{t=0}^{T-1} \frac{\pi(a_t \mid s_t)}{\mu(a_t \mid s_t)}$$

Ordinary IS estimator: $V^{\text{IS}}(s) = \frac{1}{N} \sum_{k=1}^{N} \rho^{(k)} G^{(k)}$. *Weighted IS* estimator: $V^{\text{WIS}}(s) = \frac{\sum_k \rho^{(k)} G^{(k)}}{\sum_k \rho^{(k)}}$. Ordinary IS is unbiased; weighted IS is biased but has much lower variance (especially when ρ has high variance, which happens when π and μ differ substantially). Variance of ordinary IS: can be infinite when ρ is unbounded.

Bias-variance spectrum. The n-step return: $G_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n})$.

Estimator	Bias	Variance
TD(0) $n = 1$	$O(\gamma \, \|V - V^\pi\|)$	Low (1 step)
n-step	$O(\gamma^n \, \|V - V^\pi\|)$	Medium
MC ($n = \infty$)	0	High ($\Theta(H^2 \operatorname{Var}[r])$)

Bias decreases geometrically with n; variance increases roughly linearly. The optimal n trades these off depending on the approximation quality of V and the reward variance.

Monte Carlo estimation. In the Monte Carlo (MC) approach, the agent runs complete episodes and estimates $V^\pi(s)$ by averaging the returns observed from s:

$$V^\pi(s) \approx \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_t^{(i)}$$

where $N(s)$ counts visits to s and $G_t^{(i)}$ is the return following the i-th visit. *First-visit MC* counts only the first time s appears in an episode; *every-visit MC* counts all appearances. First-visit MC is unbiased; every-visit MC has smaller variance but introduces a small bias from non-independence.

Temporal Difference learning. TD(0) bootstraps: after one step (s_t, r_t, s_{t+1}):

$$V(s_t) \leftarrow V(s_t) + \alpha \underbrace{\left[r_t + \gamma V(s_{t+1}) - V(s_t) \right]}_{\delta_t \text{ (TD error)}}$$

The TD error δ_t is a one-step look-ahead correction. Unlike MC, TD does not wait for the episode to end — it can be applied online and to continuing tasks.

Bias-variance analysis.

- **MC**: unbiased estimate of $V^\pi(s)$ since $\mathbb{E}[G_t | s_t = s] = V^\pi(s)$. High variance because G_t aggregates many stochastic steps.
- **TD**: biased (uses bootstrap estimate $V(s_{t+1})$ instead of true $V^\pi(s_{t+1})$). Low variance because only one random step is included.

The bias-variance trade-off is resolved by the n-step return:

$$G_t^{(n)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n})$$

$n = 1$ recovers TD(0); $n = \infty$ recovers MC. The λ-return mixes all n: $G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$.

Example — Random Walk: MC vs TD comparison

Five-state random walk: $\{A, B, C, D, E\}$ with $V(A) = 0$ (absorbing, reward 0) and $V(E) = 1$ (absorbing, reward 1). True values: $V(B) = \frac{1}{5}, V(C) = \frac{2}{5}, V(D) = \frac{3}{5}$. Starting from all estimates $\hat{V} = 0.5$:

- **TD(0)**: after each step, updates toward immediate reward + bootstrapped next value. Converges smoothly to true values within 20 episodes.
- **MC**: waits until episode ends, then updates all visited states with the final return. More noisy per episode but each update is an unbiased sample.

Empirically (Sutton 1988): TD(0) converges faster than MC in the random walk setting, with RMS error ≈ 0.05 vs ≈ 0.08 after 100 episodes (batch setting, $\alpha = 0.05$).

91.2 TD(0): One-Step Prediction

The simplest TD algorithm, **TD(0)**, updates after every step:

$$V(s_t) \leftarrow V(s_t) + \alpha\, \delta_t = V(s_t) + \alpha\, [r_t + \gamma V(s_{t+1}) - V(s_t)]$$

TD(0) convergence

For a fixed policy π, with learning rate α_t satisfying $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$ (Robbins-Monro conditions), TD(0) converges with probability 1 to V^π.

Bias-variance tradeoff. The TD target $r_t + \gamma V(s_{t+1})$ is biased (because V is an approximation) but low variance (it depends on only one step of randomness). MC is unbiased but high variance. This tradeoff is governed by the n-step return:

$$G_t^{(n)} \triangleq \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n})$$

$n = 1$ gives TD(0); $n \to \infty$ gives MC. **TD(λ)** interpolates via eligibility traces (see §91.4).

Comparison to gradient descent. TD(0) is NOT a true gradient descent method. Consider the MSE objective $\mathcal{L}(\theta) = \mathbb{E}_\pi[(V^\pi(s) - V_\theta(s))^2]$. The true gradient would use targets $V^\pi(s)$, which are unknown. TD uses the biased target $r + \gamma V_\theta(s')$, and the update $\Delta\theta = \alpha \delta_t \nabla_\theta V_\theta(s_t)$ treats the target as fixed (hence *semi-gradient*). This means TD does not converge to the minimum of any fixed objective, but to a different point (the *TD fixed point*) determined by the feature geometry.

The TD fixed point (linear FA). With linear approximation $V_\theta(s) = \square(s)^\top \square$, TD(0) converges to $\square^*$ satisfying:

$$\mathbf{A}\square^* = \mathbf{b}, \qquad \mathbf{A} = \mathbb{E}\big[\square(s_t)(\square(s_t) - \gamma \square(s_{t+1}))^\top\big], \qquad \mathbf{b} = \mathbb{E}[r_t \square(s_t)]$$

This is not the solution to $\min_\theta \mathbb{E}[(V^\pi(s) - V_\theta(s))^2]$ in general. The TD fixed point satisfies $\Pi \mathcal{T}^\pi V_{\theta^*} = V_{\theta^*}$ where Π is orthogonal projection onto the linear function space under the weighted norm $\|\cdot\|_\mu$.

Convergence rate and Robbins-Monro conditions. Step sizes $\{\alpha_t\}$ must satisfy: $\sum_{t=0}^{\infty} \alpha_t = \infty$ (ensures learning does not stop prematurely) and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$ (ensures noise averages out). Common schedule: $\alpha_t = \alpha_0/(1 + t\beta)$.

Example — 5-state random walk: TD(0) update sequence

States $\{1, 2, 3, 4, 5\}$, terminal states 0 (reward -1) and 6 (reward $+1$). Policy: random walk ($p = 0.5$ each direction). True values: $V^\pi(s) = (s-0)/(6-0) - 1 = s/6 - 1$ ... more precisely $V^\pi(s) = 2s/6 - 1 = (s-3)/3$. Actually $V^\pi(s) = (2s-6)/6$ so $V^\pi(3) = 0$, $V^\pi(1) = -2/3$, $V^\pi(5) = 2/3$.

Initialise: $V_0 = [0.5, 0.5, 0.5, 0.5, 0.5]$ for states $1, \ldots, 5$. $\alpha = 0.1$, $\gamma = 1$.
Episode: trajectory $3 \to 4 \to 5 \to 6$(terminal), rewards $[0, 0, +1]$.
Step $t = 0$: $s = 3, r = 0, s' = 4$. $\delta = 0 + V(4) - V(3) = 0.5 - 0.5 = 0$. No update.
Step $t = 1$: $s = 4, r = 0, s' = 5$. $\delta = 0 + V(5) - V(4) = 0$. No update. Step $t = 2$: $s = 5, r = +1, s' = 6$(terminal). $\delta = 1 + 0 - V(5) = 0.5$. $V(5) \leftarrow 0.5 + 0.1 \times 0.5 = 0.55$.
After many episodes, values converge toward $[-0.67, -0.33, 0, 0.33, 0.67]$.

Batch TD. Given a fixed dataset $\mathcal{D} = \{(s_t, r_t, s_{t+1})\}$, batch TD repeatedly applies updates over all transitions until convergence. This converges to the unique TD fixed point for the dataset, which is not the same as the Monte Carlo solution (least-squares fit to observed returns). The batch TD solution minimises $\|\Pi \mathcal{T}^\pi V_\theta - V_\theta\|_\mu^2$ over θ; the MC solution minimises $\mathbb{E}[(G_t - V_\theta(s_t))^2]$.

91.3 SARSA and Q-Learning

Control requires estimating Q rather than V (since we cannot act greedily with respect to V without knowing the model).

SARSA (on-policy TD control):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

The name comes from the quintuple $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$. SARSA is on-policy: it evaluates and improves the same policy being used to generate data.

Q-Learning (off-policy TD control, Watkins 1989):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

Q-Learning is off-policy: the target uses $\max_{a'}$ (the greedy policy) regardless of what action the agent actually took. This decoupling means Q-Learning learns Q^* directly.

Theorem: Q-Learning convergence

For a finite MDP, with all state-action pairs visited infinitely often and step sizes satisfying Robbins-Monro conditions, Q-Learning converges with probability 1 to Q^*.

Example — SARSA vs. Q-Learning on the Cliff Walk

Consider a grid with a cliff. SARSA (on-policy, ε-greedy) learns a safe path away from the cliff because it accounts for exploratory ε-greedy actions that might fall off. Q-Learning converges to the optimal (shorter) path along the cliff edge, but achieves lower online reward during training because its greedy target ignores exploration risk. This illustrates the on-policy/off-policy distinction concretely.

SARSA convergence theorem.

Theorem (SARSA convergence)

For a finite MDP, if all state-action pairs are visited infinitely often (guaranteed by e.g. ε-greedy with $\varepsilon_t \to 0$ and $\sum_t \varepsilon_t = \infty$), and step sizes satisfy Robbins-Monro conditions, then SARSA converges with probability 1 to Q^*.

The proof uses the stochastic approximation theorem (Robbins & Monro, 1951): the ODE $\dot{Q}(s,a) = \mathbb{E}[\delta_t \mid s_t = s, a_t = a]$ has Q^* as a globally asymptotically stable equilibrium, and the noise term satisfies martingale conditions.

Expected SARSA. Instead of using the sampled next action a_{t+1}, average over all next actions:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_t + \gamma \sum_{a'} \pi(a' \mid s_{t+1}) Q(s_{t+1},a') - Q(s_t,a_t)\right]$$

Expected SARSA has lower variance than SARSA (averaging over next action eliminates one source of noise) and is strictly better in expectation. It can also be used off-policy without importance sampling (unlike SARSA) if the target policy π is known.

Double Q-Learning. Standard Q-Learning overestimates because it selects and evaluates the max action using the same Q estimate: $\mathbb{E}[\max_a Q(s',a)] \geq \max_a \mathbb{E}[Q(s',a)]$. Double Q-Learning maintains two independent estimators Q_1, Q_2, updated alternately:

$$Q_1(s_t,a_t) \leftarrow Q_1(s_t,a_t) + \alpha\left[r_t + \gamma Q_2\left(s_{t+1}, \arg\max_{a'} Q_1(s_{t+1},a')\right) - Q_1(s_t,a_t)\right]$$

The action is selected by Q_1 but evaluated by Q_2, removing the upward bias. Converges to Q^* under the same conditions.

Example — Cliff walk: SARSA vs Q-Learning tabular comparison

Grid: 4×12, agent at $(0,0)$, goal at $(0,11)$, cliff at $(0,1)$–$(0,10)$ (reward -100, reset to start). All other steps: reward -1. $\gamma = 1$, $\varepsilon = 0.1$.

Optimal path: along the cliff edge (row 0), length 11, total reward -11. **Safe path**: go up to row 3, traverse, come down; length ≈ 25, total reward ≈ -25.
After 500 episodes:

Algorithm	Avg. reward (per episode)	Learned policy
Q-Learning	≈ -13 online	Optimal (cliff edge)
SARSA	≈ -17 online	Safe (row 3)

Q-Learning converges to the optimal policy but suffers more during training due to random falls. SARSA learns a risk-averse policy accounting for ε-greedy exploration. With $\varepsilon \to 0$, both would converge to the optimal policy.

91.4 Eligibility Traces and TD(lambda)

The λ-return. Interpolate between all n-step returns with geometric weighting:

$$G_t^\lambda \triangleq (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}, \quad \lambda \in [0,1]$$

$\lambda = 0$ recovers TD(0); $\lambda = 1$ recovers MC.

Eligibility traces allow the λ-return to be computed online. An eligibility trace $e_t(s)$ tracks the *credit* assigned to state s:

$$e_t(s) = \begin{cases} \gamma\lambda\, e_{t-1}(s) + 1 & \text{if } s = s_t \\ \gamma\lambda\, e_{t-1}(s) & \text{otherwise} \end{cases}$$

The TD(λ) update then applies the TD error δ_t to all states proportionally:

$$V(s) \leftarrow V(s) + \alpha\, \delta_t\, e_t(s) \quad \forall s$$

The trace decays geometrically: states visited long ago receive less credit. This provides an efficient, online, forward-backward equivalence: the TD(λ) backward-view is mathematically equivalent to the λ-return forward-view.

Forward-backward equivalence proof. The λ-return: $G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$. TD($\lambda$) backward view update: $V(s) \leftarrow V(s) + \alpha\delta_t e_t(s)$. These are equivalent (for offline/batch updates) because:

$$\sum_{t=0}^{T-1} \alpha\delta_t e_t(s) = \alpha \sum_{t=0}^{T-1} \delta_t \sum_{k\le t} (\gamma\lambda)^{t-k} \mathbf{1}[s_k = s] = \alpha \sum_{k:s_k=s} (G_k^\lambda - V(s_k))$$

This algebraic identity shows the backward-view accumulates contributions of each δ_t to all past states with exponential decay — exactly computing the λ-return update.

Watkins's Q(λ). Extend Q-Learning to eligibility traces:

$$e_t(s,a) = \begin{cases} \gamma\lambda\, e_{t-1}(s,a) + 1 & \text{if } s = s_t, a = a_t \text{ and } a_t = \arg\max_{a'} Q(s_t, a') \\ \gamma\lambda\, e_{t-1}(s,a) & \text{if } s = s_t, a = a_t \text{ and } a_t \neq \arg\max_{a'} Q(s_t, a') \\ 0 & \text{if } s = s_t, a \neq a_t \end{cases}$$

The trace is cut (set to 0) when a non-greedy action is taken, because the off-policy action breaks the connection to the λ-return for Q^*. Despite this, Watkins's Q(λ) converges to Q^* under the same conditions as Q-Learning.

Replacing vs. accumulating traces. *Accumulating*: $e_t(s) \leftarrow \gamma\lambda\, e_{t-1}(s) + \mathbf{1}[s = s_t]$. Can grow unboundedly for frequently visited states. *Replacing*: $e_t(s) \leftarrow \begin{cases} 1 & s = s_t \\ \gamma\lambda\, e_{t-1}(s) & s \neq s_t \end{cases}$. Caps the trace at 1 — prevents numerical issues and often improves performance empirically.

True online TD(λ). The exact online implementation of the forward view (van Seijen et al., 2016):

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\delta_t e_t + \alpha(Q(s_t,a_t) - Q_{\text{old}}(s_t,a_t))(e_t - 1)$$

where Q_{old} is the value before the current update. Uses "Dutch traces": $e_t = \gamma\lambda\, e_{t-1} + (1 - \alpha\gamma\lambda(\varphi_{t-1}^\top e_{t-1}))\varphi_t$ (for linear FA). True online TD(λ) is exactly equivalent to the forward view online.

Example — 5-state random walk: lambda comparison

True values: $V^\pi(s) = (s-3)/3$ for $s = 1, \ldots, 5$. After 50 episodes, RMS error $\|V - V^\pi\|_2$:

λ	RMS Error
0 (TD)	0.18
0.3	0.13
0.5	0.11
0.7	0.10
0.9	0.12
1.0 (MC)	0.19

The optimal $\lambda \approx 0.7$ balances bias-variance tradeoff for this task with $\alpha = 0.05$.

Eligibility traces. An eligibility trace $e_t(s) \in \mathbb{R}_{\geq 0}$ tracks which states are "eligible" for credit assignment. The accumulating trace updates:

$$e_t(s) = \begin{cases} \gamma\lambda\, e_{t-1}(s) + 1 & \text{if } s = s_t \\ \gamma\lambda\, e_{t-1}(s) & \text{otherwise} \end{cases}$$

The replacing trace alternative: $e_t(s_t) = 1$ (reset rather than accumulate), which prevents runaway updates on frequently visited states.

TD(λ) update rule. After every step, update all states:

$$V(s) \leftarrow V(s) + \alpha \delta_t e_t(s), \quad \forall s$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the TD error.

Forward-backward view equivalence

The *forward view* defines the λ-return target. The *backward view* implements it online via eligibility traces. Sutton (1988) proved: for the offline (batch) case with constant α, the two views produce *identical* weight updates summed over a complete episode. For online TD(λ) the equivalence is only approximate but becomes exact in the limit.

Watkins's Q(λ). For control, the trace is cut whenever a non-greedy action is taken:

$$e_t(s,a) \leftarrow \begin{cases} \gamma\lambda\, e_{t-1}(s,a) + 1 & \text{if } s = s_t, a = a_t \\ 0 & \text{if } a_t \neq \arg\max_{a'} Q(s_t, a') \text{ (non-greedy)} \\ \gamma\lambda\, e_{t-1}(s,a) & \text{otherwise} \end{cases}$$

Cutting the trace at non-greedy actions maintains the off-policy property of Q-learning. Peng's Q(λ) is a softer alternative that does not cut traces, accepting some off-policy bias.

Example — TD(λ) on 4-state chain

States $\{1,2,3,4\}$, terminal reward $+1$ at state 4. $\gamma = 0.9, \lambda = 0.8, \alpha = 0.1$. After visiting trajectory $1 \to 2 \to 3 \to 4$: TD error at step 3 is $\delta_3 = 1 - V(3)$. Traces: $e_3(3) = 1,\ e_3(2) = 0.72,\ e_3(1) = 0.518$. Updates: $\Delta V(3) = 0.1\delta_3,\ \Delta V(2) = 0.072\delta_3,\ \Delta V(1) = 0.0518\delta_3$. With $\lambda = 0$: only $V(3)$ updates. With $\lambda = 1$: all states get credit in proportion to geometric decay — equivalent to MC on this deterministic episode.

Part 92

Deep Q-Networks and Value-Based Deep RL

Combining Q-Learning with deep neural networks yields powerful agents capable of

learning from high-dimensional observations. We derive the DQN loss, analyse instabilities, and present the key stabilisation techniques: experience replay and target networks.

92.1 Function Approximation in RL

When $|\mathcal{S}|$ is large or continuous, tabular methods are infeasible. We approximate $Q(s,a) \approx Q(s,a;\square)$ with a parametric function (e.g. a neural network with weights $\square$).

The semi-gradient TD update. Minimise the *Mean Squared Bellman Error*:

$$\mathcal{L}(\square) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s',a';\square) - Q(s,a;\square)\right)^2\right]$$

The gradient update treats the TD target as a constant (hence *semi-gradient*):

$$\square \leftarrow \square + \alpha\, \delta_t\, \nabla_{\square} Q(s_t,a_t;\square)$$

where $\delta_t = r_t + \gamma \max_{a'} Q(s_{t+1},a';\square) - Q(s_t,a_t;\square)$.

Instability sources. Naively combining TD with neural networks is unstable for three reasons:

1. **Correlations**: consecutive (s_t,a_t,r_t,s_{t+1}) tuples are highly correlated, violating the i.i.d. assumption of SGD.
2. **Non-stationarity of target**: the TD target depends on $\square$, so it shifts as $\square$ updates — creating a "moving goalpost".
3. **Deadly triad**: off-policy training + function approximation + bootstrapping can diverge (Tsitsiklis & Van Roy, 1997).

Linear function approximation. With features $\square(s,a) \in \mathbb{R}^d$ and weights $\square \in \mathbb{R}^d$: $Q(s,a;\square) = \square(s,a)^\top \square$. The semi-gradient update: $\square \leftarrow \square + \alpha\delta_t \square(s_t,a_t)$. For on-policy TD with linear approximation, $\square$ converges to $\square^*$ satisfying: $\mathbf{A}\square^* = \mathbf{b}$ where:

$$\mathbf{A} = \mathbb{E}\left[\square_{s,a}\left(\square_{s,a} - \gamma \sum_{s'} \mathcal{P}(s'|s,a) \sum_{a'} \pi(a'|s')\square_{s',a'}\right)^\top\right], \quad \mathbf{b} = \mathbb{E}\big[\mathcal{R}(s,a)\,\square_{s,a}\big]$$

The matrix $\mathbf{A}$ is positive definite under the on-policy distribution, ensuring a unique solution.

Tile coding. A practical feature construction for continuous states. Divide each dimension into n tiles; overlay m tilings offset by $1/(mn)$. Each tiling produces one binary feature (whether the state falls in that tile). Total features: $m \cdot n^d$ for d dimensions. Advantages: (1) generalise within tiles; (2) no interaction between distant states; (3) $O(m)$ non-zero features (sparse, efficient). Key parameter: tile width controls generalisation bandwidth.

Radial basis functions. Features $\phi_i(s) = \exp(-\|s - c_i\|^2/(2\sigma_i^2))$ for centres $\{c_i\}$ and widths $\{\sigma_i\}$. Smooth generalisation; sensitive to hyperparameter choice.

The deadly triad diagnostic. Three conditions that together cause divergence:

1. **Function approximation**: $Q(s,a) \approx Q_\theta(s,a)$ is a neural network or other parametric approximator.

2. **Bootstrapping**: the update target depends on Q_θ (TD methods), not on observed returns (MC).

3. **Off-policy**: the distribution of (s, a) pairs used for updates differs from the on-policy distribution d^π.

Any two alone are safe (Tsitsiklis & Van Roy 1997); all three together can diverge. DQN alleviates the issue via experience replay (making the update distribution more i.i.d.) and the target network (reducing non-stationarity of the target).

Mean Squared Value Error. The standard measure of approximation quality:

$$\overline{\mathrm{VE}}(\square) = \sum_{s \in \mathcal{S}} d^\pi(s) \left(V^\pi(s) - V_\square(s)\right)^2$$

where $d^\pi(s)$ is the on-policy state distribution. Gradient descent on $\overline{\mathrm{VE}}$ gives the true gradient TD update; semi-gradient TD converges to a point within $1/(1-\gamma)$ of the $\overline{\mathrm{VE}}$ minimum.

92.2 DQN: Experience Replay and Target Networks

DQN (Mnih et al., 2015) introduces two key stabilisations:

1. Experience replay. Store transitions (s_t, a_t, r_t, s_{t+1}) in a replay buffer $\mathcal{D}$ of capacity N. At each update, sample a minibatch $\mathcal{B} \sim \mathcal{D}$ uniformly at random:

$$\mathcal{L}(\square) = \frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s') \in \mathcal{B}} \left(r + \gamma \max_{a'} Q(s', a'; \square^-) - Q(s, a; \square) \right)^2$$

This breaks temporal correlations and reuses data efficiently.

2. Target network. Use a separate *target network* $Q(\cdot; \square^-)$ with parameters $\square^-$ that are periodically copied from $\square$ (every C steps). The loss becomes:

$$y_j = r_j + \gamma \max_{a'} Q(s'_j, a'; \square^-)$$

$$\mathcal{L}(\square) = \mathbb{E}_{\mathcal{B}}\left[\left(y_j - Q(s_j, a_j; \square) \right)^2 \right]$$

Fixing $\square^-$ for C steps makes the target stationary within each period.

DQN Algorithm (simplified)

Initialise $\square$ randomly; set $\square^- \leftarrow \square$; initialise replay buffer $\mathcal{D}$.
For each episode:

1. Observe s_t; select $a_t = \arg\max_a Q(s_t, a; \square)$ with prob. $1 - \varepsilon$, else random.
2. Execute a_t, observe r_t, s_{t+1}; store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$.
3. Sample minibatch $\mathcal{B}$ from $\mathcal{D}$; compute targets y_j; update $\square$ via SGD on $\mathcal{L}(\square)$.
4. Every C steps: $\square^- \leftarrow \square$.

The instability problem. Naive Q-learning with function approximation suffers from two pathologies: (i) correlations in the sequence of observations (s_t, a_t, r_t, s_{t+1}) violate the i.i.d. assumption of SGD; (ii) the target $r + \gamma \max_{a'} Q(s', a'; \theta)$ is non-stationary because θ changes every step, creating a moving target.

Experience Replay. DQN stores transitions (s_t, a_t, r_t, s_{t+1}) in a replay buffer $\mathcal{D}$ of size $|\mathcal{D}| = 10^6$. At each training step, a minibatch of size 32 is sampled uniformly: $\{(s_j, a_j, r_j, s'_j)\}_{j=1}^{32} \sim \mathrm{Uniform}(\mathcal{D})$.

Benefits: (a) decorrelates samples; (b) each transition is used multiple times (data efficiency); (c) smooths out non-stationary data distributions.

Target Network. A separate network $Q(s, a; \theta^-)$ with frozen parameters θ^- computes the TD target:

$$y_j = r_j + \gamma \max_{a'} Q(s'_j, a'; \theta^-)$$

The parameters θ^- are updated every $C = 10{,}000$ steps: $\theta^- \leftarrow \theta$. The main network minimises:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\Big[\big(y_j - Q(s, a; \theta)\big)^2\Big]$$

DQN Algorithm (Mnih et al., 2015)

Hyperparameters: replay buffer size $|\mathcal{D}| = 10^6$; batch size 32; target update $C = 10^4$; ε-greedy with ε annealed $1.0 \to 0.1$ over 10^6 steps; $\gamma = 0.99$; RMSProp with lr=2.5×10^{-4}. **Architecture**: 3 conv layers (32@8×8, 64@4×4, 64@3×3) + 2 FC layers (512 units). Input: 4 stacked, grayscale, 84×84 frames.

Prioritised Experience Replay (PER). Instead of uniform sampling, sample transitions with probability $P(j) \propto |\delta_j|^\alpha + \epsilon$ where $|\delta_j|$ is the last TD error. High-error transitions are sampled more often, improving learning speed. Importance-sampling weights $w_j = (N \cdot P(j))^{-\beta}$ (annealed $\beta : 0.4 \to 1$) correct for the sampling bias.

92.3 Double DQN and Dueling Architecture

Overestimation bias in Q-Learning. The $\max_{a'}$ in the TD target introduces a positive bias: $\mathbb{E}[\max_i X_i] \geq \max_i \mathbb{E}[X_i]$. Over many updates, this accumulates and degrades performance.

Double DQN (van Hasselt et al., 2016) decouples action selection and evaluation:

$$y_j = r_j + \gamma Q\left(s'_j,\ \arg\max_{a'} Q(s'_j, a'; \square);\ \square^-\right)$$

The online network $\square$ selects the action; the target network $\square^-$ evaluates it. This removes the systematic overestimation.

Dueling architecture (Wang et al., 2016). Decompose Q into value and advantage streams:

$$Q(s, a; \square, \alpha, \beta) = V(s; \square, \beta) + \left(A(s, a; \square, \alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s, a'; \square, \alpha)\right)$$

The mean subtraction ensures identifiability (otherwise V and A are not separately identified). The value stream learns state quality independently of action selection, improving sample efficiency when many actions have similar values.

Formal proof of overestimation bias. Let $X_1, \ldots, X_K$ be independent unbiased estimators of $\mu_1, \ldots, \mu_K$ with variance σ^2. Then:

$$\mathbb{E}\Big[\max_i X_i\Big] \geq \max_i \mathbb{E}[X_i] = \max_i \mu_i = \mu^*$$

The inequality is strict unless all $X_i = \mu_i$ a.s. The bias grows with K and σ^2: for Gaussian $X_i \sim \mathcal{N}(\mu_i, \sigma^2)$, the overestimation is $\Theta(\sigma\sqrt{\log K})$. In Q-Learning, $K = |\mathcal{A}|$ and σ^2 is the variance of Q estimates — both can be large.

C51: Distributional RL. Instead of estimating $\mathbb{E}[G \mid s, a]$, learn the full distribution $Z(s, a)$. Represent the return distribution on a fixed support $\{z_1, \dots, z_N\}$:

$$Z(s, a) = z_i \text{ with probability } p_i(s, a; \square), \quad \sum_i p_i = 1$$

The distributional Bellman operator is:

$$(\mathcal{T}Z)(s, a) \overset{d}{=} r(s, a) + \gamma Z(s', a^*), \quad a^* = \arg\max_{a'} \mathbb{E}[Z(s', a')]$$

The loss is the cross-entropy between the projected target distribution and the predicted distribution. Advantage: captures return multimodality; the mean of the distribution gives Q^*.

NoisyNets. Replace deterministic linear layers with noisy counterparts: $y = (\mu^w + \sigma^w \odot \varepsilon^w)x + (\mu^b + \sigma^b \odot \varepsilon^b)$ where $\varepsilon^w, \varepsilon^b$ are noise variables. Parameters μ, σ are learned; ε is sampled per forward pass (or per episode for factorised noise). This enables state-dependent exploration: the network learns *how much* to explore. The noise effectively replaces ε-greedy, with the network adapting its exploration to the problem structure.

Rainbow. Combines DQN + Double + Dueling + Prioritised Replay + Multi-step returns + Distributional + NoisyNets. The combined agent achieves superhuman performance on Atari with $\sim 7\times$ better sample efficiency than DQN. Each component contributes independently; ablation studies show distributional RL and prioritised replay contribute the most.

Dueling identifiability constraint

The decomposition $Q = V + A$ is not unique (adding any function of s to V and subtracting it from A gives the same Q). The mean-centering: $Q(s, a) = V(s) + (A(s, a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s, a'))$ enforces identifiability because $\sum_a Q(s, a) = |\mathcal{A}|V(s)$, making V determined by the average Q value and A determined by deviations from that average.

PART 93

Policy Gradient Methods

Rather than learning value functions and deriving policies implicitly, policy gradient methods directly parametrise and optimise the policy. We derive the Policy Gradient Theorem from first principles and build toward REINFORCE and actor-critic algorithms.

93.1 The Policy Gradient Theorem

Direct policy parametrisation. Let $\pi_\square(a \mid s)$ be a parametrised policy. Define the *performance objective*:

$$J(\square) \triangleq \mathbb{E}_{\tau \sim \pi_\square}\left[\sum_{t=0}^{T} \gamma^t r_t\right] = \mathbb{E}_{s_0}[V^{\pi_\square}(s_0)]$$

We want $\nabla_\square J(\square)$ to perform gradient ascent.

Policy Gradient Theorem (Sutton et al., 2000)

$$\nabla_\square J(\square) = \mathbb{E}_{\tau \sim \pi_\square}\left[\sum_{t=0}^{T} \nabla_\square \log \pi_\square(a_t \mid s_t) \cdot Q^{\pi_\square}(s_t, a_t)\right]$$

Proof sketch. We use the *log-derivative trick*: $\nabla_\square \pi_\square(a \mid s) = \pi_\square(a \mid s)\, \nabla_\square \log \pi_\square(a \mid s)$. Starting from $J(\square) = \sum_\tau p_\square(\tau)\, R(\tau)$:

$$\nabla_\square J = \sum_\tau \nabla_\square p_\square(\tau)\, R(\tau) = \sum_\tau p_\square(\tau)\, \nabla_\square \log p_\square(\tau)\, R(\tau) = \mathbb{E}_{\pi_\square}[\nabla_\square \log p_\square(\tau)\, R(\tau)]$$

The trajectory log-probability decomposes as $\log p_\square(\tau) = \sum_t \log \pi_\square(a_t \mid s_t) + \sum_t \log \mathcal{P}(s_{t+1} \mid s_t, a_t)$. The transition terms do not depend on $\square$ and drop out, leaving the stated result. □

Key insight. The theorem is remarkable: it gives an exact gradient of the performance objective without differentiating through the environment dynamics $\mathcal{P}$. The gradient is an expectation over trajectories, estimable by sampling.

Full proof of the Policy Gradient Theorem. The trajectory probability under π_θ factorises as:

$$p_\theta(\tau) = \mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t)\, \mathcal{P}(s_{t+1} \mid s_t, a_t)$$

Therefore:

$$\log p_\theta(\tau) = \log \mu(s_0) + \sum_t \log \pi_\theta(a_t \mid s_t) + \sum_t \log \mathcal{P}(s_{t+1} \mid s_t, a_t)$$

The gradient with respect to θ kills all terms not containing θ:

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

Therefore:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)\, \nabla_\theta \log p_\theta(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta}\left[R(\tau) \sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right]$$

Using the *causality trick*: action a_t only affects future rewards, so $r_{t'}$ for $t' < t$ has zero correlation with $\nabla_\theta \log \pi_\theta(a_t \mid s_t)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} \right] = \mathbb{E}_\tau \left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t \right]$$

Replacing G_t with its expectation $Q^{\pi_\theta}(s_t, a_t)$ completes the theorem. □

The Fisher information matrix. The *Fisher information matrix* of the policy is:

$$\mathbf{F}(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid s)\, (\nabla_\theta \log \pi_\theta(a \mid s))^\top] = -\mathbb{E}_{\pi_\theta}[\nabla_\theta^2 \log \pi_\theta(a \mid s)]$$

$\mathbf{F}$ defines a Riemannian metric on the space of probability distributions (the *Fisher-Rao metric*). The natural policy gradient is:

$$\tilde{\nabla}_\theta J = \mathbf{F}(\theta)^{-1} \nabla_\theta J(\theta)$$

which is the steepest ascent direction in the metric space induced by $\mathbf{F}$, i.e. in the space of policies (KL divergence geometry) rather than Euclidean parameter space.

Example — Finite difference gradient verification

Policy: softmax over 2 actions, $\pi_\theta(a=1|s) = \sigma(\theta) = 1/(1+e^{-\theta})$. Scalar θ. Environment: single state, $r=1$ for $a=1$, $r=0$ for $a=0$. $J(\theta) = \pi_\theta(1|s) = \sigma(\theta)$. **Analytic**: $\nabla_\theta J = \sigma(\theta)(1-\sigma(\theta))$. **Score function**: $\mathbb{E}[r\nabla_\theta \log \pi] = 1\cdot\sigma(\theta)\cdot(1-\sigma(\theta)) + 0\cdot(-\sigma(\theta)\cdot\ldots) = \sigma(1-\sigma)$. ✓ At $\theta = 0$: both give 0.25.

93.2 REINFORCE and Variance Reduction

REINFORCE (Williams, 1992) is the simplest instantiation. Using the actual return G_t as an unbiased estimate of $Q^{\pi_\square}(s_t, a_t)$:

$$\nabla_\square J(\square) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} \nabla_\square \log \pi_\square(a_t^{(i)} \mid s_t^{(i)}) \cdot G_t^{(i)}$$

The baseline. Adding a state-dependent baseline $b(s_t)$ to the return does not bias the gradient (because $\mathbb{E}_{a\sim\pi}[\nabla_\square \log \pi(a \mid s)] = 0$) but reduces variance:

$$\nabla_\square J(\square) = \mathbb{E}\left[\sum_t \nabla_\square \log \pi_\square(a_t \mid s_t)\cdot(G_t - b(s_t))\right]$$

Proof that baseline is unbiased:

$$\mathbb{E}_{a\sim\pi}[\nabla_\square \log \pi(a \mid s)\, b(s)] = b(s)\,\mathbb{E}_{a\sim\pi}[\nabla_\square \log \pi(a \mid s)] = b(s)\cdot\nabla_\square \underbrace{\sum_a \pi(a \mid s)}_{=1} = 0$$

The optimal baseline (minimising variance) is:

$$b^*(s) = \frac{\mathbb{E}\left[\|\nabla_\square \log \pi\|^2 \cdot Q(s,a)\right]}{\mathbb{E}\left[\|\nabla_\square \log \pi\|^2\right]}$$

In practice, $b(s_t) = V_\square(s_t)$ (a learned value function) is used, leading to the advantage:

$$A^\pi(s_t, a_t) = G_t - V_\square(s_t)$$

Variance of REINFORCE. The variance of the REINFORCE estimator scales as:

$$\mathrm{Var}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t\right] = O(H^2\,\mathrm{Var}[r]/N)$$

where H is the episode length and N is the number of sampled trajectories. The H^2 factor arises because returns involve sums of H rewards, and the gradient sums H score functions. This is the fundamental limitation of Monte Carlo policy gradient: long episodes lead to extremely high variance.

Reward-to-go (causality reduction). Using $G_t^{\text{to-go}} = \sum_{k=t}^{T-1}\gamma^{k-t} r_k$ instead of G_0 reduces variance because it removes rewards before time t (which are independent of a_t):

$$\mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_0\right] = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t^{\text{to-go}}\right]$$

but $\mathrm{Var}[G_t^{\text{to-go}}] < \mathrm{Var}[G_0]$ in general. This is the *reward-to-go* trick.

Optimal baseline derivation. The variance of $\hat{g}_\theta = \sum_t f_t(G_t - b)$ where $f_t = \nabla_\theta \log \pi_\theta(a_t|s_t)$:

$$\mathrm{Var}[\hat{g}_\theta] = \mathbb{E}[\hat{g}_\theta^2] - (\mathbb{E}[\hat{g}_\theta])^2 = \mathbb{E}[\hat{g}_\theta^2] - (\nabla J)^2$$

Minimising $\mathbb{E}[\hat{g}_\theta^2]$ over scalar b (for a single step):

$$\mathbb{E}[(f(G-b))^2] = \mathbb{E}[f^2 G^2] - 2b\,\mathbb{E}[f^2 G] + b^2 \mathbb{E}[f^2]$$

Setting derivative to zero: $b^* = \mathbb{E}[f^2 G]/\mathbb{E}[f^2]$ — the baseline is the f^2-weighted average of returns. In practice, a simple $b(s) = V_\phi(s)$ is used, which approximates this optimally as long as $V_\phi \approx V^\pi$.

Control variates. In general, for any function $g(x)$ with known $\mathbb{E}[g(x)] = \mu_g$, the control variate estimator of $\mathbb{E}[f(x)]$ is: $\hat{\mu}_f = f(x) - c^*(g(x) - \mu_g)$, where $c^* = \mathrm{Cov}(f, g)/\mathrm{Var}(g)$. The baseline trick is a special case: $g(x) = \nabla \log \pi$, $\mu_g = 0$.

93.3 Actor-Critic Algorithms

Actor-critic methods maintain two networks:

- **Actor**: $\pi_\square(a \mid s)$, updated by policy gradient.
- **Critic**: $V_\square(s)$, updated by TD learning to estimate $V^{\pi_\square}$.

The critic's TD error serves as an advantage estimate:

$$\delta_t = r_t + \gamma V_\square(s_{t+1}) - V_\square(s_t) \approx A^{\pi_\square}(s_t, a_t)$$

The actor update becomes:

$$\square \leftarrow \square + \alpha_\theta\, \delta_t\, \nabla_\square \log \pi_\square(a_t \mid s_t)$$

The critic update:

$$\square \leftarrow \square - \alpha_\phi\, \delta_t\, \nabla_\square V_\square(s_t)$$

This is the **A2C** (Advantage Actor-Critic) algorithm. The one-step TD advantage is biased but low-variance; n-step returns or generalised advantage estimation (GAE, §94.2) reduce bias at modest variance cost.

Two-timescale stochastic approximation. Actor-critic stability requires the critic to converge faster than the actor: $\alpha_\phi \gg \alpha_\theta$. Formally, we require $\alpha_\theta/\alpha_\phi \to 0$ as $t \to \infty$. Then the critic sees a nearly static policy (converges to V^{π_θ}) while the actor sees a near-optimal critic (computes near-exact advantage). This *two-timescale* argument (Borkar, 2008) ensures global convergence to a locally optimal policy.

A3C (Asynchronous Advantage Actor-Critic). Multiple worker threads run in parallel, each with its own environment instance:

1. Each worker accumulates gradients for $t_{\max}$ steps (or until terminal).
2. Worker applies accumulated gradient to shared (global) parameters.
3. Worker copies global parameters and continues.

Gradients are applied asynchronously without locking. This (1) breaks temporal correlations (workers at different states), (2) improves wall-clock time (parallelism), (3) is effectively an off-policy method (stale parameters). The n-step return used:

$$G_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V_\phi(s_{t+n})$$

Compatible function approximation. For the critic $Q_w(s,a) = w^\top \square(s,a)$ to produce an unbiased policy gradient estimate, the features must satisfy the *compatibility condition*:

$$\square(s,a) = \nabla_\theta \log \pi_\theta(a \mid s)$$

Under this condition: $\sum_a \pi_\theta(a|s) Q_w(s,a) = 0$ for all s (the baseline condition is satisfied automatically), and the critic gradient equals the true policy gradient. In practice, this restricts the critic to linear approximation; modern methods use neural critics without this constraint, accepting some bias.

Critic error propagation. If the critic has approximation error ε_c (so $Q_\phi(s,a) = Q^\pi(s,a) + \varepsilon_c(s,a)$), then the actor gradient estimate has bias $\mathbb{E}[\sum_t \nabla \log \pi \cdot \varepsilon_c]$. For this bias to vanish, either $\varepsilon_c \to 0$ (critic converges) or the error is uncorrelated with the score function (unlikely in practice). This is why the two-timescale argument is essential: it ensures the critic is sufficiently accurate before the actor moves.

Part 94

Proximal Policy Optimisation

TRPO and PPO constrain the policy update to prevent destructive large steps. We derive the surrogate objective, the clipping mechanism, and generalised advantage estimation in full mathematical detail. PPO is the workhorse of modern LLM alignment.

94.1 Trust Region Policy Optimisation

The problem with naïve policy gradient. Large gradient steps can cause the policy to "fall off a cliff": a bad update collapses performance, and because the policy now generates different data, recovery is slow. We need to bound how much the policy changes per update.

The performance difference lemma (Kakade & Langford, 2002). For any two policies π and $\tilde{\pi}$:

$$J(\tilde{\pi}) - J(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^{\tilde{\pi}}, a \sim \tilde{\pi}}[A^\pi(s,a)]$$

where $d^{\tilde{\pi}}(s) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t P(s_t = s \mid \tilde{\pi})$ is the discounted state visitation measure.

The difficulty is that $d^{\tilde{\pi}}$ depends on $\tilde{\pi}$ (unknown during optimisation). TRPO approximates it with d^π and adds a KL constraint:

$$\tilde{\pi} = \arg\max_{\pi'} \; \mathbb{E}_{s \sim d^\pi, a \sim \pi}\left[\frac{\pi'(a \mid s)}{\pi(a \mid s)} A^\pi(s,a)\right] \quad \text{s.t. } \mathbb{E}_s[D_{\mathrm{KL}}(\pi(\cdot \mid s) \,\|\, \pi'(\cdot \mid s))] \le \delta$$

Full proof of the Performance Difference Lemma. For any two policies π and $\tilde{\pi}$:

$$\begin{aligned} J(\tilde{\pi}) - J(\pi) &= \mathbb{E}_{\tilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] - J(\pi) \\ &= \mathbb{E}_{\tilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t (r_t + V^{\pi}(s_{t+1})\gamma - V^{\pi}(s_t)) + V^{\pi}(s_0)\right] - J(\pi) \\ &= \mathbb{E}_{\tilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t A^{\pi}(s_t, a_t)\right] + \mathbb{E}_{\tilde{\pi}}[V^{\pi}(s_0)] - J(\pi) \end{aligned}$$

Using $J(\pi) = \mathbb{E}_{\pi}[V^{\pi}(s_0)] = \mathbb{E}_{\tilde{\pi}}[V^{\pi}(s_0)]$ (both start in same initial state):

$$J(\tilde{\pi}) - J(\pi) = \mathbb{E}_{\tilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t A^{\pi}(s_t, a_t)\right] = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\tilde{\pi}}, a\sim\tilde{\pi}}[A^{\pi}(s,a)]$$

The last equality uses the definition $d^{\tilde{\pi}}(s) = (1-\gamma)\sum_t \gamma^t P(s_t = s|\tilde{\pi})$. □

TRPO lower bound (monotonic improvement). Replacing $d^{\tilde{\pi}}$ with d^{π} in the performance difference:

$$L^{\pi}(\tilde{\pi}) \triangleq J(\pi) + \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi}, a\sim\tilde{\pi}}[A^{\pi}(s,a)]$$

The approximation error is bounded (Kakade & Langford, 2002):

$$J(\tilde{\pi}) \geq L^{\pi}(\tilde{\pi}) - \frac{4r_{\max}\gamma}{(1-\gamma)^2}\max_s D_{\mathrm{KL}}(\pi(\cdot|s) \,\|\, \tilde{\pi}(\cdot|s))$$

Denoting $\varepsilon_{\mathrm{KL}} = \max_s D_{\mathrm{KL}}(\pi(\cdot|s) \,\|\, \tilde{\pi}(\cdot|s))$, maximising the right side over $\tilde{\pi}$ with constraint $\varepsilon_{\mathrm{KL}} \leq \delta$ guarantees $J(\tilde{\pi}) \geq J(\pi)$.

Practical TRPO. The constrained optimisation: $\max_{\theta'} L^{\pi}_{\theta'}(\theta)$ s.t. $\bar{D}_{KL}(\theta, \theta') \leq \delta$ is solved by: (1) Computing $\nabla_\theta L$ and the Fisher-vector product $\mathbf{Fv}$ via Pearlmutter's trick: $\mathbf{Fv} \approx \mathbb{E}[\nabla \log \pi \cdot (\nabla \log \pi)^\top \mathbf{v}]$ (no need to form $\mathbf{F}$ explicitly); (2) Conjugate gradient to find $\mathbf{F}^{-1}\nabla L$ (the natural gradient direction); (3) Line search to find the largest step satisfying the KL constraint. Computational cost: $O(d)$ conjugate gradient steps, each $O(d)$ with Fisher-vector products.

94.2 Generalised Advantage Estimation

GAE (Schulman et al., 2016) provides a low-variance, controllable-bias advantage estimator. Define the k-step TD residuals:

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$$

The GAE advantage is the exponentially weighted sum of these residuals:

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+l}^V$$

For $\lambda = 0$: $\hat{A}_t = \delta_t^V$ (one-step TD, high bias, low variance). For $\lambda = 1$: $\hat{A}_t = \sum_l \gamma^l \delta_{t+l}^V = G_t - V(s_t)$ (MC, low bias, high variance). $\lambda \in (0,1)$ interpolates smoothly.

Example — GAE in practice

With $\gamma = 0.99$, $\lambda = 0.95$, rollout length $T = 128$: compute δ_t^V for $t = 0, \ldots, 127$, then accumulate backwards:

$$\hat{A}_{T-1} = \delta_{T-1}^V, \qquad \hat{A}_t = \delta_t^V + 0.99 \times 0.95 \times \hat{A}_{t+1}$$

This single backward pass computes all advantage estimates in $O(T)$ time.

GAE as geometric mixture. The GAE advantage at time t is:

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V$$

where $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$. This is a geometric mixture of n-step advantage estimators:

$$\hat{A}_t^{\text{GAE}} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{A}_t^{(n)}$$

where $\hat{A}_t^{(n)} = \sum_{l=0}^{n-1} (\gamma)^l \delta_{t+l}^V$ is the n-step advantage. Proof: $(1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} \sum_{l=0}^{n-1} \gamma^l \delta_{t+l} = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l} (1-\lambda) \sum_{n=l+1}^{\infty} \lambda^{n-1} = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l} \lambda^l = \sum_l (\gamma\lambda)^l \delta_{t+l}$. ✓

Bias-variance analysis. The estimator $\hat{A}_t^{\text{GAE}(\gamma,\lambda)}$ has:

- **Bias**: $O(((\gamma\lambda)^L \|V - V^\pi\|)$ where L is the effective horizon $1/(1-\gamma\lambda)$. As $\lambda \to 1$: bias $\to 0$ (no bootstrapping).
- **Variance**: grows as $O(1/(1-\gamma\lambda)^2 \cdot \text{Var}[r])$. As $\lambda \to 1$: variance $\to \infty$ (like MC).

Typical choice $\lambda = 0.95$, $\gamma = 0.99$: effective horizon $= 1/(1 - 0.99 \times 0.95) \approx 21$ steps.

V-trace (off-policy GAE). For off-policy data from behaviour policy μ, importance-sample correction:

$$\hat{A}_t^{\text{V-trace}} = \sum_{l=0}^{n-1} \gamma^l \left(\prod_{k=0}^{l-1} c_{t+k} \right) \delta_{t+l}^V$$

where $c_t = \min(\bar{c}, \pi(a_t|s_t)/\mu(a_t|s_t))$ (clipped IS ratio) and $\rho_t = \min(\bar{\rho}, \cdot)$ is similarly clipped for the TD error. Clipping prevents high-variance corrections from dominating. V-trace is used in IMPALA (Importance Weighted Actor-Learner Architecture) for massively parallel RL.

Example — GAE backward computation

Rollout: $T = 5$ steps. TD residuals $\delta = [0.5, -0.2, 0.8, 0.1, 0.3]$. $\gamma = 0.99$, $\lambda = 0.95$, so $\gamma\lambda = 0.9405$.
Backward pass: $\hat{A}_4 = 0.3$; $\hat{A}_3 = 0.1 + 0.9405 \times 0.3 = 0.382$; $\hat{A}_2 = 0.8 + 0.9405 \times 0.382 = 1.159$; $\hat{A}_1 = -0.2 + 0.9405 \times 1.159 = 0.890$; $\hat{A}_0 = 0.5 + 0.9405 \times 0.890 = 1.337$.
Total compute: 5 multiply-add operations. Contrast with forward view: $O(T^2/2)$ operations.

Motivation. The policy gradient $\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s) \cdot A^\pi(s,a)]$ requires estimating the advantage $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$. The quality of this estimate directly controls the variance of policy gradient updates.

Generalised Advantage Estimation (GAE). For a trajectory segment of length T, define $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$ (TD residuals). GAE is:

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} \triangleq \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V$$

For $\lambda = 0$: $\hat{A}_t = \delta_t$ (one-step TD advantage, low variance, high bias). For $\lambda = 1$: $\hat{A}_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V(s_t)$ (MC advantage, unbiased, high variance). For $0 < \lambda < 1$: exponentially-weighted mix, controlled bias-variance trade-off.

Practical implementation. Compute backwards over the rollout buffer:

$$\hat{A}_T = \delta_T, \qquad \hat{A}_t = \delta_t + \gamma\lambda\, \hat{A}_{t+1}$$

This $O(T)$ backward pass is the standard implementation in PPO, A3C, and related algorithms.

Example — GAE sensitivity to λ

On a simple navigation task with dense rewards, $\lambda = 0.95, \gamma = 0.99$ is a near-universal empirical default (Schulman et al., 2016). The intuition: $\lambda \approx 1$ is needed so that advantage estimates carry information about rewards several steps ahead, but $\lambda < 1$ prevents compounding variance from long-range traces.
For $T = 20$ rollout steps with $\lambda = 0.95, \gamma = 0.99$: The effective horizon of the trace $= 1/(1-\gamma\lambda) = 1/(1-0.9405) \approx 16.8$ steps, meaning the GAE advantage incorporates about 17 future TD residuals.

Value function loss. The critic is trained simultaneously to minimise:

$$\mathcal{L}^{\mathrm{VF}}(\phi) = \frac{1}{T} \sum_{t=1}^{T} \left(V_\phi(s_t) - \hat{V}_t \right)^2$$

where $\hat{V}_t = \hat{A}_t + V_\phi(s_t)$ are the GAE-estimated value targets.

94.3 PPO: Clipped Surrogate Objective

PPO-Clip (Schulman et al., 2017) replaces the KL constraint with a clipped surrogate:

$$\mathcal{L}^{\mathrm{CLIP}}(\square) = \mathbb{E}_t \left[\min\left(r_t(\square)\, \hat{A}_t,\ \mathrm{clip}(r_t(\square), 1-\varepsilon, 1+\varepsilon)\, \hat{A}_t \right) \right]$$

where the probability ratio is:

$$r_t(\square) = \frac{\pi_{\square}(a_t \mid s_t)}{\pi_{\square_{\mathrm{old}}}(a_t \mid s_t)}$$

Mechanics of the clip.

- If $\hat{A}_t > 0$ (action was better than expected): the objective increases with r_t, but is clipped at $1+\varepsilon$, preventing too large an increase in action probability.
- If $\hat{A}_t < 0$ (action was worse than expected): the objective decreases as r_t decreases, but is clipped at $1-\varepsilon$, preventing too large a decrease.

The min ensures the surrogate is a lower bound on the unclipped objective: it is pessimistic whenever the policy moves too far.

Full PPO objective includes value function and entropy terms:

$$\mathcal{L}^{\text{PPO}}(\square) = \mathbb{E}_t[\mathcal{L}_t^{\text{CLIP}} - c_1\mathcal{L}_t^{\text{VF}} + c_2\, S[\pi_\square](s_t)]$$

where $\mathcal{L}_t^{\text{VF}} = (V_\square(s_t) - V_t^{\text{targ}})^2$ is the value loss and $S[\pi_\square](s_t) = -\sum_a \pi_\square(a \mid s_t)\log\pi_\square(a \mid s_t)$ is the entropy bonus (promoting exploration). Typical values: $\varepsilon = 0.2$, $c_1 = 0.5$, $c_2 = 0.01$.

> **Connection to RLHF.** In RLHF for LLMs (Vol. V §62), the policy $\pi_\square$ is the language model, states are token prefixes, actions are the next token, and rewards come from a learned reward model. The PPO objective is applied with an additional KL penalty to prevent divergence from the reference model: $\mathcal{R}(s,a) \leftarrow r_\phi(s,a) - \beta\log(\pi_\square(a \mid s)/\pi_{\text{ref}}(a \mid s))$, where β controls the regularisation strength.

Mathematical analysis of the PPO clip. Define $r(\theta) = \pi_\theta(a|s)/\pi_{\theta_{\text{old}}}(a|s)$. The clipped objective for a single transition:

$$\mathcal{L}_t^{\text{CLIP}}(\theta) = \min\Big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\Big)$$

Case $\hat{A}_t > 0$ (good action): $r \mapsto \min(r\hat{A}_t, \text{clip}(r, 1-\varepsilon, 1+\varepsilon)\hat{A}_t)$. For $r \le 1+\varepsilon$: both terms equal $r\hat{A}_t$; gradient is $\hat{A}_t\nabla r$. For $r > 1+\varepsilon$: clipped term is $(1+\varepsilon)\hat{A}_t$, unclipped is $r\hat{A}_t > (1+\varepsilon)\hat{A}_t$. Min selects $(1+\varepsilon)\hat{A}_t$: zero gradient. Policy cannot increase action probability beyond $(1+\varepsilon)$ factor.

Case $\hat{A}_t < 0$ (bad action): For $r \ge 1-\varepsilon$: gradient is $\hat{A}_t\nabla r < 0$ (decreasing action probability). For $r < 1-\varepsilon$: clipped at $(1-\varepsilon)\hat{A}_t$; min selects unclipped $r\hat{A}_t > (1-\varepsilon)\hat{A}_t$ since $\hat{A}_t < 0$ and $r < 1-\varepsilon$... More precisely: $r\hat{A}_t$ (more negative) $< (1-\varepsilon)\hat{A}_t$; min = $r\hat{A}_t$, gradient = $\hat{A}_t\nabla r$. This continues to decrease r (make the bad action even less likely), which is correct — no clip needed on the lower side for negative advantages.

RLHF: full mathematical derivation. The RLHF pipeline (Ouyang et al., 2022):

Step 1 — Reward model. Train $r_\phi(x,y)$ using the Bradley-Terry preference model: $P(y_w \succ y_l \mid x) = \sigma(r_\phi(x,y_w) - r_\phi(x,y_l))$. Loss:

$$\mathcal{L}_{\text{RM}}(\phi) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\big[\log\sigma(r_\phi(x,y_w) - r_\phi(x,y_l))\big]$$

Step 2 — PPO with KL penalty. Augment reward with KL regularisation:

$$\tilde{r}(x,y) = r_\phi(x,y) - \beta\log\frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$$

Optimise with PPO: $\max_\theta \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(\cdot|x)}[\tilde{r}(x,y)]$. The KL penalty prevents mode collapse and preserves language model coherence. The optimal policy is $\pi^*(y|x) \propto \pi_{\text{ref}}(y|x)\exp(r_\phi(x,y)/\beta)$ — a softmax over the reward.

DPO (Direct Preference Optimisation). Bypasses explicit RL by deriving a closed-form loss: recognising that the optimal RLHF policy satisfies $\log\pi^*(y|x)/\pi_{\text{ref}}(y|x) = r(x,y)/\beta$, one can express the reward in terms of the policy and substitute into the Bradley-Terry model:

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)\right]$$

This trains the LM directly on preference data without a separate reward model or RL loop.

PART 95

Advanced Policy Optimisation: SAC, DDPG, and Continuous Control

Continuous action spaces require off-policy actor-critic methods. We derive Soft Actor-Critic through the maximum entropy framework, and present DDPG as a deterministic policy gradient algorithm, with full mathematical treatment of the reparametrisation trick.

95.1 The Maximum Entropy Framework

Standard RL maximises expected return. **Maximum Entropy RL** augments the objective with a policy entropy term:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho^\pi}[r(s_t,a_t) + \alpha\,\mathcal{H}(\pi(\cdot \mid s_t))]$$

where $\mathcal{H}(\pi(\cdot \mid s)) = -\sum_a \pi(a \mid s)\log\pi(a \mid s)$ and $\alpha > 0$ is the temperature.

Benefits: (1) the agent is incentivised to explore all high-reward actions equally, rather than collapsing to a single mode; (2) the optimal policy is soft (stochastic), providing implicit regularisation; (3) the algorithm is more robust to hyperparameter choices.

Soft value functions. The entropy-augmented Bellman equations become:

$$V^\pi_{\text{soft}}(s) = \mathbb{E}_{a\sim\pi}[Q^\pi_{\text{soft}}(s,a) - \alpha\log\pi(a \mid s)]$$

$$Q^\pi_{\text{soft}}(s,a) = r(s,a) + \gamma\,\mathbb{E}_{s'}[V^\pi_{\text{soft}}(s')]$$

Maximum entropy optimal policy derivation. The max-entropy objective: $J_{\text{MaxEnt}}(\pi) = \mathbb{E}_\pi\left[\sum_t r(s_t,a_t) + \alpha\,\mathcal{H}(\pi(\cdot|s_t))\right]$. The soft Bellman equations: $Q^\pi_{\text{soft}}(s,a) = r(s,a)+\gamma\mathbb{E}_{s'}[V^\pi_{\text{soft}}(s')]$ where $V^\pi_{\text{soft}}(s) = \mathbb{E}_{a\sim\pi}[Q^\pi_{\text{soft}}(s,a)] + \alpha\,\mathcal{H}(\pi(\cdot|s))$.

The optimal soft policy maximises $V^\pi_{\text{soft}}(s) = \mathbb{E}_{a\sim\pi}[Q^*(s,a) - \alpha\log\pi(a|s)]$. This is the KL minimisation: $\min_\pi \mathbb{E}_s[D_{\text{KL}}(\pi(\cdot|s) \,\|\, \exp(Q^*(s,\cdot)/\alpha)/Z(s))]$. Setting gradient to zero: $\log\pi^*(a|s) = Q^*(s,a)/\alpha - \log Z(s)$, giving:

$$\pi^*(a|s) = \frac{\exp(Q^*(s,a)/\alpha)}{Z(s)}, \quad Z(s) = \int \exp(Q^*(s,a)/\alpha)\,da$$

This is a *Boltzmann distribution* over Q-values with temperature α. The soft value function: $V^*(s) = \alpha\log Z(s) = \alpha\log\int\exp(Q^*(s,a)/\alpha)\,da$ (*soft max*).

Temperature limits. $\alpha \to 0$: $\pi^*(a|s) \to \delta_{a^*}$ where $a^* = \arg\max_a Q^*(s,a)$ (standard RL). $\alpha \to \infty$: $\pi^*(a|s) \to \text{Uniform}(\mathcal{A})$ (maximum exploration).

Connection to energy-based models. Viewing $-Q^*(s,a)$ as an energy function, the optimal policy is a Gibbs distribution: $\pi^*(a|s) = \exp(-E(s,a)/\alpha)/Z(s)$. This connects maximum entropy RL to score matching, diffusion models, and energy-based policies.

EXAMPLE — 3-state MDP: soft optimal policy

$\mathcal{S} = \{s_1, s_2\}$ (non-terminal), $\mathcal{A} = \{a_1, a_2\}$. Let $Q^*(s_1,a_1) = 5$, $Q^*(s_1,a_2) = 3$. $\alpha = 1.0$: $Z = e^5 + e^3 = 148.4 + 20.1 = 168.5$. $\pi^*(a_1|s_1) = 148.4/168.5 = 0.881$; $\pi^*(a_2|s_1) = 0.119$. $\alpha = 0.5$: $Z = e^{10} + e^6 = 22026 + 403.4 = 22429$. $\pi^*(a_1|s_1) = 22026/22429 = 0.982$; much more concentrated.

$\alpha = 2.0$: $Z = e^{2.5} + e^{1.5} = 12.18 + 4.48 = 16.66$. $\pi^*(a_1|s_1) = 12.18/16.66 = 0.731$; more spread out.

Maximum Entropy RL. Standard RL maximises $\mathbb{E}[\sum_t \gamma^t r_t]$. The *maximum entropy* (*MaxEnt*) framework augments this with a policy entropy bonus:

$$J_{\text{MaxEnt}}(\pi) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}\left[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))\right]$$

where $\alpha > 0$ is the *temperature* (entropy coefficient) and $\mathcal{H}(\pi(\cdot|s)) = -\sum_a \pi(a|s) \log \pi(a|s)$.

Soft Bellman equations. Under MaxEnt, the soft value functions satisfy:

$$V^{\pi}_{\text{soft}}(s) = \mathbb{E}_{a\sim\pi}\left[Q^{\pi}_{\text{soft}}(s,a) - \alpha \log \pi(a|s)\right]$$

$$Q^{\pi}_{\text{soft}}(s,a) = r(s,a) + \gamma\, \mathbb{E}_{s'\sim P}\left[V^{\pi}_{\text{soft}}(s')\right]$$

The optimal soft policy is:

$$\pi^*(a|s) = \exp\left(\frac{Q^*_{\text{soft}}(s,a) - V^*_{\text{soft}}(s)}{\alpha}\right)$$

which is a softmax (Boltzmann) policy over the Q-values.

Benefits of MaxEnt RL

(a) **Exploration**: entropy bonus encourages exploring all promising actions, not just the greedy one. The agent maintains stochasticity rather than collapsing to a deterministic policy prematurely.
(b) **Robustness**: policies trained with MaxEnt are more robust to perturbations — they learn to succeed across many action trajectories.
(c) **Compositionality**: in certain settings, MaxEnt policies can be combined to solve new tasks without retraining (Haarnoja et al., 2017).

Connection to KL control. The MaxEnt objective can be rewritten as:

$$\pi^* = \arg\min_{\pi} \mathbb{E}_{s\sim d^{\pi}}\left[D_{\text{KL}}\left(\pi(\cdot|s) \,\|\, \frac{1}{Z(s)} \exp Q^*/\alpha\right)\right]$$

showing that optimal MaxEnt policy minimises the KL divergence from the Boltzmann distribution over Q-values. This connects to variational inference, duality theory, and the free energy principle.

95.2 Soft Actor-Critic

SAC (Haarnoja et al., 2018) is an off-policy actor-critic algorithm for continuous actions using the maximum entropy framework. It maintains:

- Two critic networks $Q_{\square_1}, Q_{\square_2}$ (with target copies $\square_1^-, \square_2^-$) for the double-Q trick;
- An actor network $\pi_{\square}(a \mid s)$, parametrised as a squashed Gaussian.

Critic loss. Using the minimum of the two critics to combat overestimation:

$$y = r + \gamma\left(\min_{j=1,2} Q_{\square_j^-}(s', \tilde{a}') - \alpha \log \pi_{\square}(\tilde{a}' \mid s')\right), \quad \tilde{a}' \sim \pi_{\square}(\cdot \mid s')$$

$$\mathcal{L}(\square_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_{\square_i}(s,a) - y\right)^2\right]$$

Actor loss. The policy is updated to minimise the KL divergence from the Boltzmann distribution $\propto \exp(Q/\alpha)$, which is equivalent to:

$$\mathcal{L}(\square) = \mathbb{E}_{s\sim\mathcal{D},\,\tilde{a}\sim\pi_\square}\left[\alpha \log \pi_\square(\tilde{a} \mid s) - \min_j Q_{\square_j}(s,\tilde{a})\right]$$

The reparametrisation trick. Since $\tilde{a}$ is sampled from $\pi_\square$, the gradient with respect to $\square$ would ordinarily require the score function estimator (high variance). The reparametrisation trick avoids this: express $\tilde{a} = f_\square(\varepsilon, s)$ with $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$ (a deterministic function of noise). For a squashed Gaussian:

$$\tilde{a} = \tanh(\mu_\square(s) + \sigma_\square(s) \odot \varepsilon)\,, \quad \varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Now $\nabla_\square \tilde{a}$ is computable by backpropagation through $f_\square$, yielding low-variance gradient estimates.

Automatic temperature tuning. The temperature α can be learned by minimising:

$$\mathcal{L}(\alpha) = \mathbb{E}_{\tilde{a}\sim\pi_\square}\left[-\alpha \log \pi_\square(\tilde{a} \mid s) - \alpha\,\bar{\mathcal{H}}\right]$$

where $\bar{\mathcal{H}}$ is a target entropy (e.g. $-|\mathcal{A}|$ for continuous spaces). This adapts α so the policy maintains the desired entropy.

Full SAC algorithm. Maintain: actor $\pi_\theta(a|s)$ (squashed Gaussian), critics $Q_{\phi_1}(s,a)$, $Q_{\phi_2}(s,a)$, target critics $Q_{\bar{\phi}_1}$, $Q_{\bar{\phi}_2}$, temperature α (or fixed).

SAC pseudocode

Initialise networks and replay buffer $\mathcal{D}$.
For each environment step:

1. Sample $a_t \sim \pi_\theta(\cdot|s_t)$, observe r_t, s_{t+1}.
2. Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$.

For each gradient step:

1. Sample minibatch $\mathcal{B}$ from $\mathcal{D}$.
2. Compute target: $y = r + \gamma(\min_i Q_{\bar{\phi}_i}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s'))$ where $\tilde{a}' \sim \pi_\theta(\cdot|s')$.
3. Update critics: $\phi_i \leftarrow \phi_i - \alpha_Q \nabla_{\phi_i} \mathbb{E}_{\mathcal{B}}[(Q_{\phi_i}(s,a) - y)^2]$.
4. Update actor: $\theta \leftarrow \theta - \alpha_\pi \nabla_\theta \mathbb{E}_{s,\tilde{a}\sim\pi_\theta}[\alpha \log \pi_\theta(\tilde{a}|s) - \min_i Q_{\phi_i}(s,\tilde{a})]$.
5. Update temperature: $\alpha \leftarrow \alpha - \alpha_{\text{temp}} \nabla_\alpha \mathbb{E}_{s,\tilde{a}}[-\alpha \log \pi_\theta(\tilde{a}|s) - \alpha\bar{\mathcal{H}}]$.
6. Soft update targets: $\bar{\phi}_i \leftarrow \tau\phi_i + (1-\tau)\bar{\phi}_i$.

Squashed Gaussian log-probability. The actor outputs $\mu_\theta(s), \sigma_\theta(s)$. Sample $u \sim \mathcal{N}(\mu, \sigma^2)$, then $a = \tanh(u)$. The log-probability corrects for the change of variables:

$$\log \pi_\theta(a|s) = \log \mathcal{N}(u; \mu, \sigma^2) - \sum_{i=1}^{|\mathcal{A}|} \log(1 - \tanh^2(u_i))$$

The subtracted term $\sum_i \log(1-a_i^2)$ is the log Jacobian of tanh. This is numerically computed as $\log(1-a_i^2) = \log 4 + \log\sigma(2u_i) + \log\sigma(-2u_i) + 2u_i$ for numerical stability.

Automatic temperature tuning. The target entropy $\bar{\mathcal{H}} = -|\mathcal{A}|$ (heuristic from SAC paper). Differentiating the Lagrangian: $\mathcal{L}(\alpha) = \alpha(-\log \pi_\theta - \bar{\mathcal{H}})$, gradient: $\nabla_\alpha \mathcal{L} = -\log \pi_\theta(a|s) - \bar{\mathcal{H}}$.

When the policy entropy is below target: $-\log \pi > \bar{\mathcal{H}} \Rightarrow \nabla > 0$, so α increases (more entropy incentive). When above target: α decreases. This automatically balances exploration and exploitation.

TD3 improvements over DDPG. (1) *Clipped double-Q*: $y = r + \gamma \min(Q_{\phi_1}(s', a'), Q_{\phi_2}(s', a'))$ — reduces overestimation. (2) *Delayed policy updates*: update actor every $d = 2$ critic updates — actor sees more accurate critic. (3) *Target policy smoothing*: $a' = \text{clip}(\mu_{\bar{\theta}}(s') + \epsilon, a_{\min}, a_{\max})$ where $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma_n^2), -c, c)$ — prevents overfitting to narrow peaks in Q.

95.3 Deterministic Policy Gradient

DDPG (Silver et al., 2014; Lillicrap et al., 2016) applies the *Deterministic Policy Gradient Theorem*:

$$\nabla_{\square} J(\mu_{\square}) = \mathbb{E}_{s\sim\rho^{\mu}}\left[\nabla_{\square}\mu_{\square}(s) \cdot \nabla_a Q^{\mu}(s,a)\big|_{a=\mu_{\square}(s)}\right]$$

The gradient flows through the critic: $\nabla_{\square} Q(s, \mu_{\square}(s)) = \nabla_a Q(s,a)|_{a=\mu(s)} \cdot \nabla_{\square}\mu_{\square}(s)$ (chain rule).

DDPG uses experience replay and target networks (as in DQN), with soft target updates:

$$\square^{-} \leftarrow \tau\,\square + (1-\tau)\,\square^{-}, \quad \tau \ll 1 \quad (\text{e.g. } \tau = 0.005)$$

rather than hard copies, yielding smoother training.

Deterministic Policy Gradient Theorem (Silver et al., 2014). Let $\mu_\theta : \mathcal{S} \to \mathcal{A}$ be a deterministic policy. Define: $J(\mu_\theta) = \mathbb{E}_{s\sim\rho^{\mu}}[r(s, \mu_\theta(s))]$.

DPG Theorem

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^{\mu}}\left[\nabla_\theta \mu_\theta(s) \cdot \nabla_a Q^{\mu}(s,a)\big|_{a=\mu_\theta(s)}\right]$$

Proof sketch. $J = \int_{\mathcal{S}} \rho^{\mu}(s) Q^{\mu}(s, \mu_\theta(s))\, ds$. Taking gradient and applying chain rule: $\nabla_\theta J = \int \rho^{\mu}(s)[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu} + \nabla_\theta Q^{\mu}]\, ds$. Expanding $\nabla_\theta Q^{\mu}$ (which depends on θ through the policy governing future states) leads to a recursive expression that simplifies to the stated result via the discounted state visitation. □

Connection to stochastic PGT. The stochastic policy $\pi_\sigma(a|s) = \mathcal{N}(a; \mu_\theta(s), \sigma^2 I)$. As $\sigma \to 0$, the stochastic gradient converges to the deterministic gradient: $\lim_{\sigma\to 0} \nabla_\theta J(\pi_\sigma) = \nabla_\theta J(\mu_\theta)$. This formalises the sense in which DPG is a limiting case of SPG.

DDPG algorithm. Off-policy with exploration noise $\mathcal{N}_t$ (Ornstein-Uhlenbeck process with mean reversion): $\mathbf{n}_{t+1} = \mathbf{n}_t + \theta_{\text{OU}}(\mu_{\text{OU}} - \mathbf{n}_t)\Delta t + \sigma_{\text{OU}}\sqrt{\Delta t}\,\square$ where $\square \sim \mathcal{N}(0, I)$. The OU process generates temporally correlated noise appropriate for physical control tasks (smooth exploration trajectories). Policy in environment: $a_t = \mu_\theta(s_t) + \mathcal{N}_t$.

Soft target updates: $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$ with $\tau = 0.005$. This is smoother than hard updates (DQN style) and more stable for continuous action spaces.

Off-policy DPG with importance sampling. When the behaviour policy β differs from μ, the off-policy gradient is: $\nabla_\theta J = \mathbb{E}_{s\sim\rho^{\beta}}[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu}(s,a)|_{a=\mu(s)}]$. Remarkably, the importance weight $\rho^{\mu}(s)/\rho^{\beta}(s)$ drops out because the deterministic policy has a point mass at $\mu_\theta(s)$, so the action IS the action and no correction is needed for the action sampling. The state distribution correction is often neglected in practice (as in DDPG), which is an approximation.

PART 96

Model-Based Reinforcement Learning

Model-based RL learns an explicit dynamics model and uses it for planning. We cover Dyna-Q, world models, and the mathematics of planning in latent space, including the connection to optimal control.

96.1 Model-Based vs. Model-Free RL

The sample efficiency argument. Model-free RL (Q-Learning, PPO, SAC) requires millions of environment interactions for complex tasks. Model-based RL (MBRL) learns a dynamics model $\hat{\mathcal{P}}(s' \mid s, a)$ and $\hat{\mathcal{R}}(s, a)$, then uses it to generate *synthetic experience* — dramatically improving sample efficiency.

The bias-variance tradeoff. A learned model introduces bias: errors in $\hat{\mathcal{P}}$ compound over long rollouts. MBRL algorithms must balance between (a) longer model rollouts (more data, more bias) and (b) shorter rollouts (less bias, less data).

Model learning objectives. Given a dataset $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}$, learn $\hat{\mathcal{P}}_\psi$ and $\hat{\mathcal{R}}_\psi$ by:

- **MLE**: $\max_\psi \sum_{(s,a,r,s') \in \mathcal{D}} \log p_\psi(s' \mid s, a) + \log p_\psi(r \mid s, a)$. For Gaussian models: reduces to MSE. For discrete states: categorical cross-entropy.
- **ELBO (for stochastic latent models)**: as in DreamerV3, maximise evidence lower bound over a latent-variable model.

Compound error in multi-step rollouts. After k steps of model rollout, the distribution of visited states diverges from the true distribution: $\left\| P^k_{\hat{\mathcal{P}}}(s' \mid s_0) - P^k_{\mathcal{P}}(s' \mid s_0) \right\|_1 \leq k\varepsilon_{\text{model}}$ where $\varepsilon_{\text{model}} = \max_{s,a} \left\| \hat{\mathcal{P}}(\cdot|s,a) - \mathcal{P}(\cdot|s,a) \right\|_1$. Errors compound linearly: k-step rollout error $\leq k\varepsilon_{\text{model}}$. This motivates short rollouts (small k) for model-based planning.

MBPO (Model-Based Policy Optimisation). Branched rollouts (Janner et al., 2019): start from real states $s \in \mathcal{D}_{\text{env}}$, unroll for k steps using the model, add synthetic transitions to $\mathcal{D}_{\text{model}}$. Train policy on $\mathcal{D}_{\text{env}} \cup \mathcal{D}_{\text{model}}$ with SAC. Key result: under model error ε_m and policy error ε_π, the combined return error is: $\varepsilon_{\text{policy}} = O(k\varepsilon_m + \varepsilon_\pi)$. Optimal rollout length $k^* = O(\sqrt{\varepsilon_\pi / \varepsilon_m})$ balances model bias and policy variance.

Gaussian process world models (PILCO). Model: $s_{t+1} = f(s_t, a_t) + \varepsilon$, $f \sim \mathcal{GP}(0, k(\cdot, \cdot))$. GP gives analytic uncertainty $\sigma^2(s,a) = k(s,a) - \mathbf{k}^\top (\mathbf{K} + \sigma_\varepsilon^2 I)^{-1} \mathbf{k}$. PILCO (Deisenroth & Rasmussen, 2011) propagates state distributions analytically through the GP, computing policy gradient using the chain rule through the GP predictions. Sample efficiency: PILCO learns cartpole in ~ 30 seconds of real interaction vs. $\sim 10^6$ steps for model-free RL.

Ensemble uncertainty. Train M models $\{\hat{\mathcal{P}}^{(m)}_\psi\}$ with different random seeds. Uncertainty: $u(s,a) = \text{Var}_m[\hat{\mathcal{P}}^{(m)}_\psi(s'|s,a)]$. Use as exploration bonus or to identify unreliable model regions. Disagreement is high in unvisited regions → directs exploration.

Model-Free RL learns a policy directly from experience: Q^π or π are estimated from sampled transitions (s, a, r, s') without ever explicitly representing $\mathcal{P}(s'|s,a)$.

- *Sample efficiency*: poor — requires many real environment interactions.
- *Asymptotic performance*: strong — can find globally optimal policies given enough data.
- *Examples*: DQN, PPO, SAC, TD3.

Model-Based RL learns or is given a dynamics model $\hat{\mathcal{P}}(s'|s,a)$ and uses it for planning (computing values or policies without interacting with the real environment).

- *Sample efficiency*: high — a single real trajectory can generate many simulated ones.
- *Compounding error*: small model errors compound over multi-step rollouts.
- *Examples*: Dyna-Q, MBPO, Dreamer, AlphaZero (tree search + learned model).

The World Model

A world model $\mathcal{M} = (\hat{\mathcal{P}}, \hat{\mathcal{R}})$ consists of a learned transition function $\hat{\mathcal{P}} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ and reward function $\hat{\mathcal{R}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Model-based planning uses $\mathcal{M}$ to generate synthetic experience or compute value functions analytically via DP.

Sample efficiency comparison. On standard benchmarks (MuJoCo locomotion), MBPO (Janner et al., 2019) achieves the asymptotic performance of SAC in $5\times$ fewer environment steps by using k-step model rollouts (typically $k = 1$–5) to augment the real replay buffer.

When to prefer each.

- High sample cost (robotics, clinical trials): model-based.
- Accurate simulation already available (games): model-free.
- High-dimensional visual input, hard to model: model-free (or latent model like Dreamer).

96.2 Dyna-Q: Integrating Learning and Planning

Dyna-Q (Sutton, 1991) augments Q-Learning with a model-based planning step:

Dyna-Q Algorithm

For each real environment step (s, a, r, s'):

1. Update $Q(s,a)$ via Q-Learning (direct RL update).
2. Update model: $\hat{\mathcal{R}}(s,a) \leftarrow r$; $\hat{\mathcal{P}}(s'|s,a) \leftarrow s'$.
3. Repeat n times (planning): sample previously seen (s,a); simulate r, s' from model; update $Q(s,a)$ via Q-Learning on simulated transition.

Each planning step has the same computational cost as a real step but uses zero additional environment interactions. With $n = 50$ planning steps per real step, Dyna-Q achieves $\sim 50\times$ better sample efficiency on tabular MDPs.

Why planning steps equal real steps. Each planning step in Dyna-Q applies one Q-Learning update. The TD error in a planning step is $\delta = \hat{r} + \gamma \max_{a'} Q(s', a') - Q(s, a)$ using model-generated (s, a, r, s'). If the model is accurate ($\hat{r} \approx r$, $\hat{P}(s'|s,a) \approx P(s'|s,a)$), then the planning update is statistically equivalent to a real environment interaction. n planning steps therefore provide the same information as n real steps, at the cost of n model calls.

Prioritised sweeping. After a Q-Learning update at (s,a) with error δ, the value of predecessor states s_{prev} such that $\hat{P}(s|s_{\text{prev}}, a_{\text{prev}}) > 0$ may also change. Prioritise $(s_{\text{prev}}, a_{\text{prev}})$ by the predicted TD error $|r(s_{\text{prev}}, a_{\text{prev}}) + \gamma \max_{a'} Q(s, a') - Q(s_{\text{prev}}, a_{\text{prev}})|$. Use a priority queue; pop the highest-priority pair for each planning step. Convergence is faster because large Bellman errors propagate backwards rapidly.

Convergence of Dyna-Q.

Theorem

For a finite MDP with a deterministic environment and accurate model, with all real state-action pairs visited infinitely often and step sizes satisfying Robbins-Monro conditions, Dyna-Q converges to Q^*.

The proof combines Q-Learning convergence with the observation that planning updates on model-generated data have the same statistical properties as Q-Learning updates on real data (when the model is accurate).

Hallucinated replay. In practice: mix real transitions (from $\mathcal{D}_{\text{env}}$) and model-generated transitions. Mixing ratio f: fraction of minibatch drawn from model rollouts. MBPO uses $f = 0.95$ (mostly synthetic). As model error decreases with more data, optimal f can increase. The key insight: even a slightly inaccurate model can accelerate learning if the model-generated data distributes the value information more efficiently across state space.

Linear Dyna. In feature space $\phi(s) \in \mathbb{R}^d$: model $\hat{s}_{t+1} = \mathbf{F}\phi(s_t) + \mathbf{G}a_t + \mathbf{g}$, reward $\hat{r}_t = \mathbf{f}^\top \phi(s_t)$. Planning step: generate $(\phi(s), a, r, \phi(s'))$ using the linear model; apply semi-gradient update. Planning is purely matrix operations — no environment calls needed. Limitation: linear models cannot capture multimodal transitions or complex dynamics.

Dyna architecture. Dyna (Sutton, 1991) interleaves real experience and simulated experience from a learned model. One step of real interaction is followed by n steps of simulated planning.

Dyna-Q algorithm.

1. Observe (s_t, a_t, r_t, s_{t+1}) from real environment.
2. Direct RL update: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$.
3. Model update: $\hat{\mathcal{P}}(s_t, a_t) \leftarrow s_{t+1}$; $\hat{\mathcal{R}}(s_t, a_t) \leftarrow r_t$ (tabular model).
4. Planning: repeat n times:
   (a) Sample $(\tilde{s}, \tilde{a})$ uniformly from previously visited state-action pairs.
   (b) Simulate: $\tilde{r}, \tilde{s}' \leftarrow \hat{\mathcal{R}}(\tilde{s}, \tilde{a}), \hat{\mathcal{P}}(\tilde{s}, \tilde{a})$.
   (c) Q-update: $Q(\tilde{s}, \tilde{a}) \leftarrow Q(\tilde{s}, \tilde{a}) + \alpha[\tilde{r} + \gamma \max_{a'} Q(\tilde{s}', a') - Q(\tilde{s}, \tilde{a})]$.

With $n = 50$ planning steps per real step, Dyna-Q solves simple gridworlds in roughly $\frac{1}{50}$ the real samples of model-free Q-learning.

Example — Maze with shortcut

A 9×6 maze is used in Sutton & Barto (2018) to illustrate Dyna-Q. After 1000 steps, a shortcut is opened. Model-free Q-learning ignores the model and must rediscover the shortcut purely from new experience. Dyna-Q's model stores the old paths; after encountering the shortcut once, the planning phase rapidly propagates value through the new route, adapting in far fewer steps.

Dyna-Q+ (exploration bonus). To handle non-stationary environments, add a bonus $\kappa\sqrt{\tau(s,a)}$ to the simulated reward, where $\tau(s,a)$ is the time since (s,a) was last tried. This encourages revisiting under-explored parts of the model.

96.3 World Models and Latent Space Planning

World models learn a compact latent representation $z_t \in \mathbb{R}^d$ of the environment state and perform planning entirely in latent space:

$$z_{t+1} = f_\square(z_t, a_t) + \square, \quad \hat{r}_t = g_\square(z_t, a_t)$$

DreamerV3 (Hafner et al., 2023) learns a world model with a RSSM (Recurrent State-Space Model) and trains the actor-critic entirely on imagined rollouts in latent space:

1. **World model training**: learn encoder $q_\psi(z_t|s_t, a_{<t})$, transition $p_\psi(z_{t+1}|z_t, a_t)$, decoder $p_\psi(s_t|z_t)$, reward $p_\psi(r_t|z_t)$ by optimising the ELBO.
2. **Imagination**: unroll the transition model for H steps from encoded states; compute returns using a learned critic.
3. **Actor-critic update**: update actor and critic on imagined trajectories.

Connection to optimal control. With a known differentiable model, policy optimisation reduces to a constrained optimisation problem solvable by gradient through time (BPTT through the model). *iLQR* (iterative Linear-Quadratic Regulator) is an efficient second-order method for this:

$$\min_{\{a_t\}} \sum_{t=0}^{T-1} c(s_t, a_t) \quad \text{s.t.} \quad s_{t+1} = f(s_t, a_t)$$

iLQR iterates: (1) backward pass computing optimal local quadratic control laws; (2) forward pass applying the control and updating the trajectory. This connects RL to classical optimal control.

Recurrent State Space Model (RSSM). The RSSM (Hafner et al., 2019) factorises the world model into deterministic and stochastic components:

$$h_t = f_\psi(h_{t-1}, z_{t-1}, a_{t-1}) \quad \text{(GRU: deterministic)}$$

$$z_t \sim q_\psi(z_t \mid h_t, x_t) \quad \text{(encoder: posterior)}$$

$$\hat{z}_t \sim p_\psi(\hat{z}_t \mid h_t) \quad \text{(prior: for imagination)}$$

The ELBO training objective:

$$\mathcal{L}_\psi = \mathbb{E}_q\big[\log p_\psi(x_t \mid h_t, z_t)\big] - \beta\, D_{\mathrm{KL}}\big(q_\psi(z_t \mid h_t, x_t) \,\|\, p_\psi(z_t \mid h_t)\big)$$

The reconstruction term trains the decoder; the KL term trains the prior to match the posterior.

MuZero. Learns a model without reconstruction, purely for planning:

- *Representation*: $s = h_\theta(o)$ maps observations to latent states.
- *Dynamics*: $r, s' = g_\theta(s, a)$ predicts reward and next latent state.
- *Prediction*: $\pi, v = f_\theta(s)$ predicts policy and value from latent state.

Training signal: MCTS improves the policy target; the model is trained to be "value-equivalent" — not to reconstruct observations, but to produce the same value estimates as the real environment.

MCTS with learned model. At each decision step, MuZero runs N MCTS simulations. Each simulation: (1) *Selection*: descend tree using UCB score $u(s,a) = Q(s,a) + c\hat{\pi}(a|s)\sqrt{N(s)}/(1+N(s,a))$; (2) *Expansion*: expand leaf with prior $\hat{\pi}$ and value $\hat{v}$ from f_θ; (3) *Backup*: update $Q(s,a)$ along the path. After N simulations, the visit counts $\{N(s,a)\}$ define the improved policy target.

Latent imagination in DreamerV3. Unroll H-step trajectories using the prior dynamics $p_\psi(\hat{z}_{t+1}|\hat{h}_{t+1})$ and reward model $p_\psi(r_t|\hat{h}_t, \hat{z}_t)$, starting from encoded real states. Actor-critic trained entirely on imagined data: the actor maximises returns computed by the critic on imagined trajectories. No gradient flows through the environment.

Connection to optimal control (iLQR). For known differentiable dynamics $s_{t+1} = f(s_t, a_t)$ and cost $c(s,a)$, the gradient of total cost with respect to control sequence is computed by *backpropagation through time*: $\nabla_{a_t} J = \frac{\partial J}{\partial s_T}\frac{\partial s_T}{\partial a_t}$. This requires the product of Jacobians $\prod_{k=t}^{T-1}\frac{\partial f}{\partial s_k}$, which suffers from vanishing/exploding gradients for long horizons. iLQR avoids this via a second-order backward pass (computing V_{ss} and Q_{ss} exactly around the current trajectory), making it much more numerically stable.

PART 97

Multi-Agent Reinforcement Learning

When multiple agents interact in a shared environment, the mathematical framework extends to Markov Games. We cover Nash equilibria, cooperative and competitive settings, and the CTDE paradigm underlying modern MARL algorithms.

97.1 Markov Games

Markov Game (Stochastic Game)

A *Markov Game* with n agents is a tuple $(\mathcal{S}, \mathcal{A}_1, \dots, \mathcal{A}_n, \mathcal{P}, \mathcal{R}_1, \dots, \mathcal{R}_n, \gamma)$ where:

- $\mathcal{A}_i$ is the action space of agent i;
- $\mathcal{P}(s' \mid s, a_1, \dots, a_n)$ is the joint transition;
- $\mathcal{R}_i(s, a_1, \dots, a_n)$ is agent i's reward.

For $n = 1$: reduces to MDP. For $n = 2$, $\mathcal{R}_1 = -\mathcal{R}_2$: zero-sum game.

The non-stationarity problem. From any single agent's perspective, other agents are part of the environment — but they are learning and changing their policies simultaneously. This breaks the stationarity assumption of single-agent RL: the effective environment is non-stationary.

Game theory foundations. A Markov Game with $n = 2$ agents and $\mathcal{R}_1 = -\mathcal{R}_2$ is a *zero-sum stochastic game*. For such games, the minimax theorem generalises: the value exists and equals the minimax value. For general-sum games, the situation is more complex — values are not well-defined without specifying an equilibrium concept.

Non-stationarity in MARL. From agent i's perspective, the environment includes all other agents' policies $\{\pi_j\}_{j\neq i}$. If agent j updates its policy from π_j to π_j', the transition kernel changes: $\mathcal{P}(s'|s,a_i) = \sum_{a_{-i}}\prod_{j\neq i}\pi_j(a_j|s)\,\mathcal{P}(s'|s,a_i,a_{-i})$ depends on $\{\pi_j\}$. As all agents learn simultaneously, each faces a non-stationary environment. This invalidates standard RL convergence guarantees.

Dec-POMDP

A *Decentralised POMDP* is a tuple $(\mathcal{S}, \{A_i\}, \{\mathcal{O}_i\}, \mathcal{P}, \{R_i\}, \gamma)$ where each agent i observes $o_i \in \mathcal{O}_i$ (private) but not the global state. Solving Dec-POMDPs is NEXP-hard in general (Bernstein et al., 2002), making approximate methods essential.

Curse of multi-agent dimensionality. The joint action space $\mathcal{A} = \prod_i \mathcal{A}_i$ grows exponentially in n. For n agents each with $|A_i| = K$ actions: $|\mathcal{A}| = K^n$. This makes tabular MARL infeasible for large n; factored representations and mean-field approximations are necessary.

Mean-field approximation. When n is large and agents are exchangeable (symmetric), the joint effect of all other agents can be approximated by their empirical action distribution (mean field): $\bar{a} = \frac{1}{n-1}\sum_{j \neq i} a_j$. The Q-function factorises: $Q(s, a_i, a_{-i}) \approx Q(s, a_i, \bar{a})$. This reduces the K^n action space to $K \times K$ (per-agent action × mean action), making mean-field MARL tractable.

97.2 Nash Equilibria in Markov Games

Nash Equilibrium

A joint policy $(\pi_1^*, \ldots, \pi_n^*)$ is a *Nash Equilibrium* if for every agent i and any alternative policy π_i:

$$J_i(\pi_i^*, \pi_{-i}^*) \geq J_i(\pi_i, \pi_{-i}^*)$$

where π_{-i}^* denotes all agents' policies except i.

No agent can unilaterally improve by deviating from a Nash equilibrium. In zero-sum two-player games, the Nash equilibrium corresponds to the minimax solution:

$$\max_{\pi_1} \min_{\pi_2} J(\pi_1, \pi_2) = \min_{\pi_2} \max_{\pi_1} J(\pi_1, \pi_2)$$

Self-Play. In symmetric zero-sum games, iterating: "train agent against a copy of itself" converges toward Nash equilibrium policies. AlphaGo/AlphaZero use this combined with MCTS.

Nash equilibrium: existence. In a finite normal-form game, a Nash equilibrium in mixed strategies always exists (Nash, 1950). The proof uses Kakutani's fixed-point theorem on the best-response correspondence $B_i(\sigma_{-i}) = \arg\max_{\sigma_i} u_i(\sigma_i, \sigma_{-i})$. For zero-sum games, the equilibrium is unique (minimax theorem) and can be found by linear programming.

Linear programming for zero-sum matrix games. For a 2×2 zero-sum game with payoff matrix $\mathbf{A}$ (agent 1's payoff): $\max_p \min_q p^\top \mathbf{A} q = \min_q \max_p p^\top \mathbf{A} q$ (Von Neumann minimax theorem). LP for agent 1: $\max_{p,v} v$ s.t. $\mathbf{A}^\top p \geq v\mathbf{1}$, $p \geq 0$, $\mathbf{1}^\top p = 1$. The dual LP gives agent 2's optimal mixed strategy q^*. For general matrix games with $m \times n$ actions: LP has $m + 1$ variables and $n + 1$ constraints — polynomial time.

Self-play and convergence. In *independent Q-Learning*, each agent uses standard Q-Learning treating others as part of environment. Convergence is not guaranteed: agents may oscillate or diverge (rock-paper-scissors example: any deterministic policy is beaten by the opponent's best response). In zero-sum games with perfect information, *fictitious self-play* (each agent best-responds to the historical average strategy of opponents) converges to Nash (Robinson, 1951). *Counterfactual Regret Minimisation* (*CFR*) achieves no-regret and converges to Nash for extensive-form games — foundation of poker AI (Libratus, AlphaStar).

Regret matching. At each round t, agent chooses action proportional to positive regret: $\sigma_t(a) \propto \max(0, R_t(a))$, where $R_t(a) = \sum_{\tau<t}[u(a, \sigma_{-\tau}) - u(\sigma_\tau, \sigma_{-\tau})]$. Average strategies $\bar{\sigma}_t$ converge to a correlated equilibrium. For two-player zero-sum games: converges to Nash.

97.3 Cooperative MARL: CTDE

Centralised Training with Decentralised Execution (CTDE) is the dominant paradigm for cooperative MARL. During training, agents share information (global state, joint actions, others' observations); during execution, each agent acts only on local observations.

QMIX (Rashid et al., 2018). Factorises the joint Q-function as:

$$Q_{\text{tot}}(\mathbf{s}, \mathbf{a}) = f_{\text{mix}}(Q_1(o_1, a_1), \dots, Q_n(o_n, a_n);\, \mathbf{s})$$

where the mixing network f_{mix} has non-negative weights (ensured by absolute value activations), satisfying the *Individual-Global-Max* (*IGM*) condition: $\arg\max_{\mathbf{a}} Q_{\text{tot}} = (\arg\max_{a_1} Q_1, \dots, \arg\max_{a_n} Q_n)$. This allows centralised training while executing greedily with individual Q-networks.

VDN and QMIX comparison. *Value Decomposition Networks* (Sunehag et al., 2017): $Q_{\text{tot}} = \sum_i Q_i(o_i, a_i)$. Linear decomposition: each agent's Q-function is independent. Simple but limited expressivity.

QMIX: $Q_{\text{tot}} = f_{\text{mix}}(Q_1, \dots, Q_n; s)$ with monotone mixing (non-negative weights). The IGM condition: $\arg\max_{\mathbf{a}} Q_{\text{tot}} = (\arg\max_{a_1} Q_1, \dots, \arg\max_{a_n} Q_n)$ holds because $\partial Q_{\text{tot}}/\partial Q_i \geq 0$ ensures the joint argmax decomposes into individual argmaxes. QMIX cannot represent all cooperative Q-functions (only those satisfying monotonicity), but this restriction is often not binding in practice.

MAPPO. Multi-Agent PPO with centralised critic. Each agent runs an independent actor $\pi_{\theta_i}(a_i|o_i)$. A centralised critic $V_\phi(s)$ (using global state) computes advantages $\hat{A}_t = G_t - V_\phi(s_t)$. The PPO objective is applied independently per agent: $\mathcal{L}^i(\theta_i) = \mathbb{E}_t[\min(r_t^i \hat{A}_t^i, \text{clip}(r_t^i, 1\pm\varepsilon)\hat{A}_t^i)]$. Parameter sharing (using same θ for all agents when symmetric) greatly improves sample efficiency.

Credit assignment: COMA. The counterfactual baseline (Foerster et al., 2018) for agent i: $\hat{A}_i(s, \mathbf{a}) = Q(s, \mathbf{a}) - \sum_{a_i'} \pi_i(a_i'|o_i)\, Q(s, (a_i', \mathbf{a}_{-i}))$ measures how much better agent i's action was compared to the default policy (marginalising over a_i). This is an unbiased advantage estimator that correctly assigns credit to agent i, avoiding the problem where all agents get the same team reward regardless of individual contribution.

Communication: TarMAC. Agents broadcast messages $m_i = g_\theta(h_i)$ at each step. Agent i attends over received messages: $c_i = \sum_j \alpha_{ij} m_j$ where $\alpha_{ij} = \text{softmax}(\langle q_i, k_j \rangle)$, q_i = query (from h_i), k_j = key (from m_j). Differentiable end-to-end; agents learn what to communicate.

Part 98

Exploration, Bandits, and the Exploration-Exploitation Trade-off

The exploration-exploitation dilemma is central to RL. We treat multi-armed bandits as a clean mathematical setting, derive UCB and Thompson Sampling, and extend to posterior exploration in MDPs.

Multi-Armed Bandit (MAB)

A MAB is a sequence of decisions: at each round $t = 1, \dots, T$, the agent chooses arm $a_t \in \{1, \dots, K\}$ and receives reward $r_t \sim \mathcal{R}_{a_t}$. The goal is to maximise $\sum_{t=1}^{T} r_t$ or equivalently minimise *regret*:

$$\text{Regret}(T) = T\mu^* - \mathbb{E}\left[\sum_{t=1}^{T} r_t\right] = \sum_{i=1}^{K} \Delta_i \, \mathbb{E}[N_i(T)]$$

where $\mu^* = \max_i \mu_i$, $\Delta_i = \mu^* - \mu_i$, and $N_i(T)$ is the number of times arm i was pulled.

Lower bound. By the Lai-Robbins theorem (1985), any consistent algorithm satisfies:

$$\liminf_{T\to\infty} \frac{\text{Regret}(T)}{\log T} \geq \sum_{i:\Delta_i>0} \frac{\Delta_i}{d(\mu_i, \mu^*)}$$

where d is the KL divergence between distributions. This is a fundamental information-theoretic lower bound.

Regret decomposition. The regret decomposes as:

$$\text{Regret}(T) = T\mu^* - \mathbb{E}\left[\sum_{t=1}^{T} r_t\right] = \sum_{i} \Delta_i \mathbb{E}[N_i(T)]$$

where $\Delta_i = \mu^* - \mu_i \geq 0$ is the suboptimality gap. To minimise regret: (a) optimal arm should be pulled $\Theta(T)$ times; (b) suboptimal arms should be pulled $O(\log T)$ times.

Lai-Robbins lower bound (sketch). For any consistent algorithm ($\mathbb{E}[N_i(T)] = o(T^\alpha)$ for all $\alpha > 0$) and Bernoulli rewards:

$$\liminf_{T\to\infty} \frac{E[N_i(T)]}{\log T} \geq \frac{1}{d_{\text{KL}}(\mu_i, \mu^*)}$$

where $d_{\text{KL}}(p, q) = p \log(p/q) + (1-p)\log((1-p)/(1-q))$ is the Bernoulli KL divergence. The argument: distinguish between μ_i and μ^* requires $\Omega(\log T / d_{\text{KL}}(\mu_i, \mu^*))$ samples.

Explore-then-commit. Pull each arm m times, then commit to the empirical best. Regret: $O(Km + \sum_i \Delta_i e^{-2m\Delta_i^2})$. Optimising m: set $m = (T/K)^{2/3}$ (assuming known T). Regret: $O(K^{1/3}T^{2/3})$. Suboptimal compared to UCB's $O(\log T)$, but simpler.

Example — Beta-Bernoulli bandit: posterior update

5 arms with unknown Bernoulli means. Prior: Beta$(1, 1)$ (uniform) for all arms. After 10 pulls of arm 3 with 7 successes: posterior is Beta$(1 + 7, 1 + 3) =$ Beta$(8, 4)$. Mean $= 8/(8 + 4) = 0.667$. Variance $= 8 \cdot 4/(12^2 \cdot 13) = 0.0171$.
Thompson Sampling: sample $\tilde{\mu}_i \sim$ Beta(α_i, β_i) for each arm. If sample for arm 3 is 0.72 and all others are < 0.72: pull arm 3. Over time, the arm with the highest true mean receives the most samples (posterior concentrates), and the algorithm's regret matches the Lai-Robbins bound.

Contextual bandits. At each round, observe context $x_t \in \mathcal{X}$; select action; receive $r_t = \mu(x_t, a_t) + \varepsilon_t$. Goal: compete with best policy $\pi^* : \mathcal{X} \to \mathcal{A}$ in hindsight. *LinUCB*: assume

$\mu(x, a) = \theta_a^\top x$. Confidence ellipsoid: $\hat{\theta}_a = (\mathbf{A}_a)^{-1}\mathbf{b}_a$ where $\mathbf{A}_a = \lambda I + \sum_{t:a_t=a} x_t x_t^\top$, $\mathbf{b}_a = \sum_{t:a_t=a} r_t x_t$. UCB: $a_t = \arg\max_a \hat{\theta}_a^\top x_t + \alpha\sqrt{x_t^\top \mathbf{A}_a^{-1} x_t}$. Regret bound: $O(d\sqrt{T \log T})$ where d is the feature dimension.

98.2 UCB: Upper Confidence Bounds

UCB1 (Auer et al., 2002). Select arm:

$$a_t = \arg\max_i \left[\hat{\mu}_i(t) + \sqrt{\frac{2\ln t}{N_i(t)}}\right]$$

where $\hat{\mu}_i(t)$ is the empirical mean and the second term is the *confidence bonus*. The idea: the confidence interval $[\hat{\mu}_i - c, \hat{\mu}_i + c]$ contracts as $N_i(t)$ grows; UCB1 is optimistic in the face of uncertainty.

Theorem: UCB1 regret bound

UCB1 achieves:

$$\text{Regret}(T) \leq \sum_{i:\Delta_i>0} \left[\frac{8\ln T}{\Delta_i} + \left(1 + \frac{\pi^2}{3}\right)\Delta_i\right]$$

The $O(\log T)$ regret matches the Lai-Robbins lower bound up to constants, making UCB1 asymptotically optimal.

Frequentist upper confidence bounds. The UCB1 algorithm (Auer et al., 2002) maintains sample counts n_i and empirical means $\hat{\mu}_i$ for each arm $i \in \{1, \ldots, K\}$:

$$a_t = \arg\max_i \left[\hat{\mu}_i + \sqrt{\frac{2\ln t}{n_i}}\right]$$

The second term is the exploration bonus: it is large when arm i has been pulled infrequently (small n_i) and shrinks as data accumulates.

Regret bound for UCB1.

Theorem (Auer, Cesa-Bianchi, Fischer, 2002)

UCB1 achieves expected regret:

$$\mathbb{E}[R_T] \leq \sum_{i:\Delta_i>0} \left[\frac{8\ln T}{\Delta_i} + 1 + \frac{\pi^2}{3}\right]$$

where $\Delta_i = \mu^* - \mu_i$ is the gap between the best arm and arm i. This is $O(\sqrt{KT\ln T})$ when all Δ_i are equal.

Proof sketch. Arm i is pulled at round t only if $\hat{\mu}_i + c_i(t) \geq \hat{\mu}^* + c^*(t)$. For the suboptimal arm to be selected, either $\hat{\mu}^*$ is underestimated or $\hat{\mu}_i$ is overestimated. Applying Hoeffding's inequality to each event and summing over t gives the bound. □

Example — UCB1 on 3-arm bandit

Arms with $\mu_1 = 0.2, \mu_2 = 0.5, \mu_3 = 0.8$. After $t = 100$ steps, suppose $n_1 = 5, n_2 = 15, n_3 = 60$. UCB scores: $0.2 + \sqrt{2\ln 100/5} = 0.2 + \sqrt{1.228} \approx 1.31$; $0.5 + \sqrt{2\ln 100/15} \approx$

$0.5 + 0.789 = 1.29$; $0.8 + \sqrt{2\ln 100/60} \approx 0.8 + 0.394 = 1.19$. UCB1 selects arm 1 (the worst!) because it hasn't been explored enough — exactly the intended behaviour.

UCB-V and KL-UCB.

- **UCB-V**: replaces $\sqrt{\ln t/n_i}$ with $\sqrt{2V_i \ln t/n_i} + 3b \ln t/n_i$ where V_i is the empirical variance. Tighter for low-variance arms.
- **KL-UCB**: uses $q_i^+ = \max\{q : n_i \cdot \mathrm{KL}(\hat{\mu}_i \| q) \leq \ln t\}$ as the upper bound. Asymptotically optimal for Bernoulli rewards.

98.3 Thompson Sampling

Thompson Sampling (Thompson, 1933) takes a Bayesian approach. Maintain a posterior $p(\mu_i \mid \text{data})$ over arm means; at each round, sample $\tilde{\mu}_i \sim p(\mu_i \mid \text{data})$ for each arm, then pull $a_t = \arg\max_i \tilde{\mu}_i$.

For Bernoulli rewards with Beta prior: $p(\mu_i) = \text{Beta}(\alpha_i, \beta_i)$. After pulling arm i and observing $r \in \{0, 1\}$:

$$\alpha_i \leftarrow \alpha_i + r, \qquad \beta_i \leftarrow \beta_i + (1 - r)$$

The posterior is still Beta (conjugate family). Thompson Sampling achieves the Lai-Robbins lower bound and performs very well empirically.

Beta-Bernoulli conjugate model (full derivation). Prior: $\mu_i \sim \text{Beta}(\alpha_i, \beta_i)$, with density $p(\mu) \propto \mu^{\alpha-1}(1-\mu)^{\beta-1}$. Likelihood for S successes in N pulls: $p(S|N, \mu) = \binom{N}{S}\mu^S(1-\mu)^{N-S}$. Posterior by Bayes' rule: $p(\mu|S, N) \propto \mu^{\alpha+S-1}(1-\mu)^{\beta+N-S-1} = \text{Beta}(\alpha + S, \beta + N - S)$. The Beta family is conjugate to the Bernoulli likelihood. After each pull: $\alpha_i \leftarrow \alpha_i + r$ and $\beta_i \leftarrow \beta_i + (1 - r)$.

Gaussian bandit. Rewards $r \sim \mathcal{N}(\mu_i, \sigma^2)$ with known σ^2. Prior $\mu_i \sim \mathcal{N}(\mu_0, \tau_0^2)$. Posterior after n observations with mean $\bar{r}$: $\mu_i|\bar{r} \sim \mathcal{N}\left(\frac{\sigma^2\mu_0 + n\tau_0^2\bar{r}}{\sigma^2 + n\tau_0^2}, \frac{\sigma^2\tau_0^2}{\sigma^2 + n\tau_0^2}\right)$. Thompson Sampling: sample $\tilde{\mu}_i \sim$ posterior, pull $\arg\max_i \tilde{\mu}_i$.

PSRL (Posterior Sampling for RL). Extend Thompson Sampling to MDPs. At each episode k: (1) Sample MDP $\tilde{M}_k = (\hat{\mathcal{P}}_k, \hat{\mathcal{R}}_k)$ from posterior over MDPs. (2) Solve $\tilde{M}_k$ to get optimal policy $\tilde{\pi}_k^*$. (3) Execute $\tilde{\pi}_k^*$ for one episode; update posterior. PSRL achieves $\tilde{O}(\sqrt{H^2SAK})$ Bayesian regret (Osband et al., 2013), matching UCBVI. It is also robust to model misspecification and adapts automatically to problem structure.

Ensemble as approximate Thompson Sampling. Maintain M models $\{\hat{Q}_m\}$ initialised differently. At each episode, sample one model $m_k \sim \text{Uniform}(\{1, \ldots, M\})$, execute greedy policy under $\hat{Q}_{m_k}$. This is Thompson Sampling from the implicit ensemble prior. Ensemble disagreement $\text{Var}_m[\hat{Q}_m(s, a)]$ serves as an uncertainty estimate for exploration bonuses.

Example — 5-arm Beta bandit: 50 rounds

True means: $\mu = [0.2, 0.5, 0.7, 0.4, 0.3]$. Arm 3 is optimal. Round 1–10: Beta(1,1) prior, round-robin exploration. After 2 pulls each with approximate 0.7 successes for arm 3: $\alpha_3 = 2.4, \beta_3 = 0.6$, mean $= 0.80$. Other arms more uncertain. Rounds 11–30: Thompson Sampling concentrates on arm 3 but occasionally explores arm 2 (0.5). Rounds 31–50: Arm 3 pulled ≈ 30 times, total regret $\approx (0.7-0.7)\times 30 + (0.7-0.5)\times 15 + \ldots \approx 4$. Optimal pulls $\approx 50 \times 0.7 = 35$; actual ≈ 33. Near-optimal behaviour.

98.4 Exploration in MDPs

ε-greedy selects a random action with probability ε and the greedy action otherwise. Simple but inefficient: all non-greedy actions are equally likely regardless of their uncertainty.

Optimism in the face of uncertainty (OFU). Extend the UCB idea: add an exploration bonus $b(s,a) = \beta/\sqrt{N(s,a)}$ to the reward:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + b(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

States and actions visited rarely get high bonuses, incentivising exploration.

Count-based exploration extends to deep RL via *pseudo-counts* (Bellemare et al., 2016): approximate $N(s)$ using a density model $\rho_n(s)$ via:

$$\hat{n}(s) = \frac{\rho_n(s)(1-\rho'_n(s))}{\rho'_n(s) - \rho_n(s)}$$

where $\rho'_n(s)$ is the density after incrementally training on s. This connects information-theoretic exploration to practical deep RL.

Boltzmann exploration. The softmax policy: $\pi(a|s) = \exp(Q(s,a)/\tau)/\sum_{a'}\exp(Q(s,a')/\tau)$. Temperature $\tau \to 0$: greedy; $\tau \to \infty$: uniform. Annealing schedule: $\tau_t = \tau_0/(1+\beta t)$. Exponential annealing $\tau_t = \tau_0 \cdot c^t$ is used in practice. Softmax exploration assigns non-negligible probability to suboptimal actions proportional to their relative Q-values — more principled than ε-greedy.

Count-based exploration in deep RL (Bellemare et al., 2016). A density model $\rho_n : \mathcal{S} \to [0,1]$ trained on n observations. After observing state s for the $(n+1)$-th time, update to ρ_{n+1}. Pseudo-count: $\hat{n}(s) = \rho_n(s)(1-\rho_{n+1}(s))/(\rho_{n+1}(s) - \rho_n(s))$. Bonus: $r^+(s) = \beta/\sqrt{\hat{n}(s)}$. Connection to prediction gain: $\hat{n}(s) \approx 1/(2\,\mathrm{PG}(s))$ where $\mathrm{PG}(s) = \log\rho_{n+1}(s) - \log\rho_n(s)$.

RND (Random Network Distillation, Burda et al., 2019). Fix a random target network $f : \mathcal{S} \to \mathbb{R}^k$. Train a predictor network $\hat{f}_\theta : \mathcal{S} \to \mathbb{R}^k$ to predict $f(s)$ on observed states. Exploration bonus: $r^+(s) = \left\|f(s) - \hat{f}_\theta(s)\right\|^2$. For unvisited states, prediction error is high (large bonus); for frequently visited states, error is low. Training signal for predictor: MSE $= \left\|f(s_t) - \hat{f}_\theta(s_t)\right\|^2$, minimised by gradient descent. Crucially, the random network f is fixed — this avoids the moving target problem.

ICM (Intrinsic Curiosity Module, Pathak et al., 2017). Learn a feature space $\phi : \mathcal{S} \to \mathbb{R}^d$ via an inverse dynamics model: predict action a_t from $\phi(s_t), \phi(s_{t+1})$. This ensures ϕ captures task-relevant features (ignoring TV noise, uncontrollable distractors). Forward model: predict $\phi(s_{t+1})$ from $\phi(s_t), a_t$. Intrinsic reward: $r^+(s_t,a_t,s_{t+1}) = \left\|\phi(s_{t+1}) - \hat{\phi}(s_{t+1})\right\|^2$. This measures how surprising the transition is in the learned feature space.

Part 99

Theoretical Foundations: Convergence, Complexity, and Limits

We collect the theoretical guarantees of RL: PAC-MDP sample complexity, regret bounds for online RL, the deadly triad, and connections to statistical learning theory.

99.1 PAC-MDP Learning

PAC-MDP (Strehl et al., 2009)

An algorithm is *Probably Approximately Correct in MDPs* (PAC-MDP) if, with probability $\geq 1-\delta$, it takes at most $\text{poly}(|\mathcal{S}|, |\mathcal{A}|, 1/\varepsilon, 1/\delta, 1/(1-\gamma))$ steps that are *not* ε-optimal.

RMAX (Brafman & Tennenholtz, 2002) is the canonical PAC-MDP algorithm. It maintains an "unknown" set of state-action pairs and assigns them the maximum possible reward $r_{\max}$, creating an optimistic MDP. Once a pair is visited $m = O(\log(1/\delta)/\varepsilon^2)$ times, it transitions to "known". RMAX is PAC-MDP with sample complexity $\tilde{O}(|\mathcal{S}|^2|\mathcal{A}|/(\varepsilon^3(1-\gamma)^6))$.

PAC-MDP definition (formal).

PAC-MDP (Strehl et al., 2009)

An algorithm $\mathcal{A}$ is PAC-MDP if, for any $\varepsilon, \delta > 0$ and any MDP M, with probability at least $1-\delta$ over the randomness of $\mathcal{A}$ and M, the number of time steps at which $\mathcal{A}$ is not ε-optimal is at most $\mathcal{O}(\text{poly}(|\mathcal{S}|, |\mathcal{A}|, 1/\varepsilon, 1/\delta, 1/(1-\gamma)))$.

Here, a step is *not ε-optimal* if the agent takes an action such that $V^*(s_t) - V^{\pi_t}(s_t) > \varepsilon$.

RMAX algorithm. Partition (s,a) pairs into *known* (visited $\geq m$ times) and *unknown*. Maintain empirical model $\hat{\mathcal{P}}, \hat{\mathcal{R}}$ for known pairs. For unknown pairs: assign optimistic reward $r_{\max} = \max_{s,a} \mathcal{R}(s,a)/(1-\gamma)$. Solve the resulting optimistic MDP; execute the optimal policy. Sample complexity: $\tilde{O}(|\mathcal{S}|^2|\mathcal{A}|/(\varepsilon^3(1-\gamma)^6))$. Key idea: either (a) the optimistic MDP has high value and the agent gets high reward, or (b) the agent visits an unknown pair and acquires new knowledge. Either way, progress is made.

Sample complexity lower bound. Any PAC-MDP algorithm requires $\Omega(|\mathcal{S}||\mathcal{A}|/(\varepsilon^2(1-\gamma)^3))$ samples in the worst case. The lower bound comes from the observation that each (s,a) pair must be visited $\Omega(1/\varepsilon^2)$ times to estimate $\mathcal{P}(\cdot|s,a)$ and $\mathcal{R}(s,a)$ to $O(\varepsilon)$ accuracy (via Hoeffding's inequality).

Regret vs PAC. PAC bounds the total number of bad steps; regret bounds the cumulative loss. A PAC-MDP algorithm with bound $N_0(\varepsilon,\delta)$ bad steps has regret $\leq \varepsilon T + r_{\max} N_0$. Conversely, a no-regret algorithm (regret $= o(T)$) is PAC with $\varepsilon = \text{Regret}(T)/T \to 0$. For practical purposes: PAC is useful for bounded-horizon guarantees; regret is more useful for online performance.

99.2 Regret Bounds for Online RL

In the episodic setting with K episodes of length H, the *cumulative regret* is:

$$\text{Regret}(K) = \sum_{k=1}^{K} [V^*(s_1) - V^{\pi_k}(s_1)]$$

UCBVI (Azar et al., 2017) achieves near-optimal regret:

$$\text{Regret}(K) = \tilde{O}\big(\sqrt{H^2|\mathcal{S}||\mathcal{A}|K}\big)$$

This is $\tilde{O}(\sqrt{K})$ — the regret grows sublinearly, so the average regret per episode vanishes. The information-theoretic lower bound is $\Omega(\sqrt{H|\mathcal{S}||\mathcal{A}|K})$, matching up to $\sqrt{H}$ factors.

UCBVI algorithm. For episode k, state s, action a, step h:

$$\hat{V}_{k,h}(s) = \min\left(H,\ \max_a\left[\hat{\mathcal{R}}_k(s,a) + b_k(s,a,h) + \sum_{s'}\hat{\mathcal{P}}_k(s'|s,a)\hat{V}_{k,h+1}(s')\right]\right)$$

Bonus: $b_k(s,a,h) = c\sqrt{H^2\log(SAK/\delta)/N_k(s,a,h)}$. The min with H ensures bounded values; the bonus ensures optimism.

Regret proof structure. Define the "good event" $\mathcal{G}$: $\hat{\mathcal{P}}_k$ and $\hat{\mathcal{R}}_k$ lie within their confidence sets. By Hoeffding's inequality: $\mathcal{G}$ holds with probability $\geq 1-\delta$. Conditioned on $\mathcal{G}$: the optimistic estimates satisfy $\hat{V}_{k,1}(s_1) \geq V_1^*(s_1)$ for all k (optimism). The regret per episode: $V_1^*(s_1) - V_1^{\pi_k}(s_1) \leq \hat{V}_{k,1}(s_1) - V_1^{\pi_k}(s_1)$. Summing over K episodes and bounding the total bonus using the Cauchy-Schwarz inequality: $\sum_k b_k(s_{t,k},a_{t,k},h) \leq O(\sqrt{H^2SAK\log(SAK/\delta)})$. This gives the stated bound.

Lower bound matching. The information-theoretic lower bound of $\Omega(\sqrt{HSAK})$ follows from the bandit lower bound applied to a hard MDP construction (binary tree with SA leaves, reward only at root). UCBVI is minimax optimal up to $\sqrt{H}$ factors.

Reward-free exploration. (Jin et al., 2020) Decouple exploration from exploitation: Phase 1: explore for K_0 episodes without any reward signal (pure exploration). Phase 2: given any reward function r, plan using the collected data. Sample complexity: $\tilde{O}(H^5SA/\varepsilon^2)$ episodes in Phase 1; Phase 2 is computation-only. This is useful when the reward is not known at training time (e.g. multi-task RL, offline planning).

99.3 The Deadly Triad and Divergence

The *deadly triad* (Sutton & Barto, 2018) refers to the combination of three components that can cause Q-Learning with function approximation to diverge:

1. **Function approximation**: representing Q as a parametric function.
2. **Bootstrapping**: using estimates of future value in the TD target.
3. **Off-policy training**: learning from data generated by a different policy.

Baird's counterexample (1995) shows that TD with linear function approximation, bootstrapping, and off-policy data *diverges* for a simple 7-state MDP, with $\|\square_t\| \to \infty$. The key insight: semi-gradient TD does not minimise a proper objective; the update direction $\delta_t \nabla_\square Q$ is not the gradient of any fixed loss function when the target also depends on $\square$.

Gradient TD methods (Maei et al., 2009) fix this by minimising the *Projected Bellman Error* with true gradient updates. For linear $Q(s,a;\square) = \square(s,a)^\top\square$:

$$\text{PBE}(\square) = \|\Pi\mathcal{T}^\pi Q_\square - Q_\square\|_\mu^2$$

where Π is the projection onto the function class. GTD2 and TDC are convergent gradient methods for this objective, at the cost of maintaining a secondary weight vector.

Baird's counterexample (full specification). 7 states: $\{1,\dots,6,7\}$. Two actions: *dashed* (from any state, go to state 7 with prob 1) and *solid* (from state i, go to state i for $i \in \{1,\dots,6\}$; from state 7, uniform over $\{1,\dots,6\}$). Linear function approximation: $V_\theta(s) = \theta_{s+7}$ for $s = 1,\dots,6$ and $V_\theta(7) = 2\theta_7$. More precisely: features for state $i \leq 6$: $\phi_i = e_i + e_7$ (two components active); feature for state 7: $\phi_7 = 2e_7$. Behaviour policy μ: take solid action with prob $6/7$, dashed with prob $1/7$. Target policy π: always dashed.

Under off-policy TD with $\gamma = 0.99$: the weight vector θ_t diverges. The TD update at state $i \leq 6$ (solid action): $\delta = 0 + 0.99V_\theta(i) - V_\theta(i) = -0.01V_\theta(i)$. This is a contraction — but the distribution is wrong. The coupling under the off-policy distribution creates instability.

Gradient TD (GTD2). Minimise the Projected Bellman Error PBE(θ) with a secondary weight vector w:

$$w \leftarrow w + \beta_t(\delta_t\phi_t - \gamma\phi_{t+1}(w^\top\phi_t) - w) \quad \text{(inner loop, fast)}$$

$$\theta \leftarrow \theta + \alpha_t\delta_t\phi_t - \alpha_t\gamma\phi_{t+1}(\phi_t^\top w) \quad \text{(outer loop, slow)}$$

The w vector approximates the gradient of PBE; θ follows the true gradient. GTD2 converges with probability 1 for any combination of function approximation, bootstrapping, and off-policy.

TDC (TD with Gradient Correction). A variant of GTD2 with faster convergence: $\theta \leftarrow \theta + \alpha_t[\delta_t\phi_t - \gamma\phi_{t+1}(\phi_t^\top w)]$. Equivalently: standard TD update + correction term $-\alpha_t\gamma\phi_{t+1}(\phi_t^\top w)$ that removes the bias from using θ in the target.

Emphatic TD. Reweights updates by the *emphasis* M_t, which tracks how important state s_t is under the target policy's visitation: $M_t = \gamma\lambda I_t + \gamma(1-\lambda)M_{t-1}$, where I_t is an interest weight. Updates: $\theta \leftarrow \theta + \alpha_t M_t \delta_t \phi_t$. Emphatic TD converges under the deadly triad for any $\lambda > 0$.

When is the deadly triad safe? On-policy + linear FA + bootstrapping: always converges (LSTD converges to TD fixed point). Off-policy + linear FA + no bootstrapping (MC): converges (LSTSQ solution exists). Off-policy + nonlinear FA + bootstrapping: can diverge even with experience replay (e.g. unconstrained neural TD). The fix: either use on-policy methods (PPO, A3C) or convergent off-policy methods (GTD2, TDC, CQL).

99.4 Connections to Supervised Learning

The distributional shift problem. In supervised learning, train and test distributions match. In RL with function approximation, the distribution μ used for updates depends on the policy π, which changes as learning proceeds. This mismatch is a fundamental source of instability.

Offline RL (Levine et al., 2020) addresses this: learn entirely from a fixed dataset $\mathcal{D}$ without further environment interaction. The key challenge is the *distribution shift* between the data distribution $\mu_\mathcal{D}$ and the learnt policy's state visitation. Conservative methods (CQL, IQL) penalise Q-values for out-of-distribution actions:

$$\mathcal{L}^{\text{CQL}}(\square) = \alpha\mathbb{E}_{s\sim\mathcal{D}}\left[\log\sum_a \exp Q_\square(s,a) - \mathbb{E}_{a\sim\mu_\mathcal{D}(s)}[Q_\square(s,a)]\right] + \frac{1}{2}\mathcal{L}^{\text{Bellman}}(\square)$$

The first term pushes down Q-values for actions not in $\mathcal{D}$; the second maintains Bellman consistency.

CQL (Conservative Q-Learning, Kumar et al., 2020). The CQL objective penalises Q-values for actions outside the dataset:

$$\min_Q \alpha\Big(\mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q(s,a)] - \mathbb{E}_{s\sim\mathcal{D},a\sim\pi_\beta(a|s)}[Q(s,a)]\Big) + \frac{1}{2}\mathcal{L}^{\text{Bellman}}(Q)$$

where μ is a distribution over actions (e.g. uniform or trained maximiser) and π_β is the behaviour policy. The first term pushes Q down for out-of-distribution actions; the second

maintains Bellman consistency. CQL provides a lower bound on the true Q-values, which prevents overoptimistic policy extraction.

IQL (Implicit Q-Learning, Kostrikov et al., 2021). Avoids ever querying $Q(s,a)$ for OOD actions by using expectile regression: $V(s) = \mathbb{E}_\tau[\max_a Q(s,a)] \approx \text{expectile}_\tau[Q(s,a)]$ for $\tau \in (0.5, 1)$. Loss: $\mathcal{L}_V(\phi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}[L_2^\tau(Q_\psi(s,a) - V_\phi(s))]$ where $L_2^\tau(u) = |\tau - \mathbb{1}(u<0)|u^2$ (asymmetric quadratic). For $\tau \to 1$: $V(s) \to \max_a Q(s,a)$ (hard max); for $\tau = 0.5$: $V(s) = \mathbb{E}_\beta[Q(s,a)]$ (mean).

Decision Transformer. RL as sequence modelling: input the trajectory $(G_t, s_t, a_t, G_{t-1}, s_{t-1}, \ldots)$ to a GPT-style transformer, predict next action autoregressively. At inference: condition on a desired return G_{target} and current state; the transformer generates the action. Training: supervised learning on offline data, conditioning on actual returns. No Bellman equation, no value function, no policy gradient — pure sequence-to-sequence prediction.

GAIL (Generative Adversarial Imitation Learning, Ho & Ermon, 2016). Given expert demonstrations $\mathcal{D}_E$, train a discriminator $D_\phi(s,a) \in [0,1]$ to distinguish agent and expert state-action pairs: $\max_\phi \mathbb{E}_{\pi_\theta}[\log D_\phi(s,a)] + \mathbb{E}_{\mathcal{D}_E}[\log(1 - D_\phi(s,a))]$. Use $r(s,a) = -\log D_\phi(s,a)$ as reward for the agent; optimise agent with PPO. GAIL implicitly recovers the reward function consistent with expert behaviour (inverse RL), without requiring $\mathcal{P}$ or explicit reward specification.

Foundation models for RL. Pre-trained language models or vision models provide rich state representations. *In-context RL*: provide the model with a context of (s,a,r) triplets; it predicts the next action given the context (Thompson Sampling-like behaviour emerges). *Algorithm Distillation* (Laskin et al., 2022): train a transformer on RL learning histories (full sequences of episodes); it learns to implement its own RL algorithm in-context.

PART 100

Advanced Topics: Distributional RL, Offline RL, and RL for LLMs

We collect the most recent and impactful developments: distributional perspectives on value estimation, the offline RL problem in depth, and the mathematical foundations of RLHF and alignment methods for large language models.

100.1 Distributional Reinforcement Learning

The distributional Bellman operator. Instead of the scalar $Q^*(s,a) = \mathbb{E}[G|s,a]$, distributional RL models the *return distribution* $Z(s,a) \overset{d}{=} G \mid s_0 = s, a_0 = a$. The distributional Bellman operator:

$$(\mathcal{T}Z)(s,a) \overset{d}{=} r(s,a) + \gamma Z(s^*, \pi^*(s^*))$$

where $s^* \sim \mathcal{P}(\cdot|s,a)$ and $\pi^* = \arg\max_a \mathbb{E}[Z(s,a)]$. This is a contraction in the Wasserstein-p metric W_p:

$$W_p(\mathcal{T}Z_1, \mathcal{T}Z_2) \le \gamma\, W_p(Z_1, Z_2)$$

with unique fixed point Z^*.

C51 implementation. Discretise support: $\{z_{\min}, z_{\min} + \Delta z, \ldots, z_{\max}\}$ with $N = 51$ atoms, $\Delta z = (z_{\max} - z_{\min})/(N-1)$. Policy network outputs N-dimensional softmax: $p_i(s,a;\theta) = P(Z(s,a) = z_i)$.

Bellman update projects the shifted distribution onto the fixed support:

$$\hat{p}_j = \sum_{i=1}^{N} p_i(s', a^*; \theta^-) \cdot \left[1 - \frac{|[\hat{T}z_i]_{z_{\min}}^{z_{\max}} - z_j|}{\Delta z}\right]^+$$

where $\hat{T}z_i = r + \gamma z_i$ (clipped to $[z_{\min}, z_{\max}]$) and $a^* = \arg\max_a \mathbb{E}[Z(s', a; \theta^-)]$. Loss: cross-entropy $\mathcal{L}(\theta) = -\sum_j \hat{p}_j \log p_j(s, a; \theta)$.

QR-DQN (Quantile Regression DQN). Instead of fixed atoms, learn the quantile function $F^{-1}_{Z(s,a)}(\tau)$ for N quantile levels $\tau_i = (2i-1)/(2N)$. The quantile regression loss:

$$\mathcal{L}_\tau(\theta) = \mathbb{E}_{z \sim Z}[|\tau - \mathbf{1}(z < \hat{z}^\tau_\theta)| \cdot |z - \hat{z}^\tau_\theta|]$$

QR-DQN achieves tighter approximation than C51 (no projection step) and matches return distributions more faithfully.

IQN (Implicit Quantile Networks). Learn a single network $q_\theta(s, a, \tau)$ mapping arbitrary $\tau \in [0, 1]$ to quantile estimates. At each update, sample fresh $\tau \sim \text{Uniform}(0, 1)$. This gives a continuous representation of the return distribution, subsumes C51 and QR-DQN, and enables risk-sensitive policies: $\pi_\beta(s) = \arg\max_a \mathbb{E}_{\tau \sim h_\beta(\tau)}[q_\theta(s, a, \tau)]$ where $h_\beta(\tau) = \beta\tau^{\beta-1}$ is a risk measure (CVaR for $\beta < 1$).

100.2 Offline Reinforcement Learning: Mathematics of Policy Pessimism

Problem formulation. Given a fixed dataset $\mathcal{D} = \{(s, a, r, s')\}$ collected by some (possibly unknown) behaviour policy π_β, learn the best possible policy π without further environment interaction.

The distribution shift challenge. The policy learned offline may take actions $a \notin \text{supp}(\pi_\beta(\cdot|s))$ for states s. For these OOD actions, the Q-network has no training data and will typically produce overoptimistic estimates (via extrapolation errors in neural networks). Executing such a policy online leads to catastrophic failures.

Pessimism principle. Conservative offline RL (Kumar et al., 2020): construct a lower bound $\underline{Q}(s, a)$ on $Q^\pi(s, a)$. Plan with respect to $\underline{Q}$. The optimal pessimistic policy: $\pi_{\text{pessimistic}} = \arg\max_{\pi \in \Pi_\beta} \mathbb{E}_\pi[\underline{Q}(s, \pi(s))]$ where Π_β restricts to policies supported by π_β. The bound must be tight enough that the pessimistic policy is near-optimal.

CQL formal derivation. CQL minimises the following regularised Bellman error:

$$\hat{Q}^{k+1} \leftarrow \arg\min_Q \alpha \underbrace{\left(\mathbb{E}_{s \sim \mathcal{D}, a \sim \mu}[Q(s, a)] - \mathbb{E}_{(s,a) \sim \mathcal{D}}[Q(s, a)]\right)}_{\text{push down OOD, push up in-dist}} + \frac{1}{2}\mathbb{E}_{(s,a,s') \sim \mathcal{D}}\left[\left(Q(s, a) - \hat{\mathcal{B}}^\pi \hat{Q}^k(s, a)\right)^2\right]$$

where $\hat{\mathcal{B}}^\pi$ is the empirical Bellman operator and μ is a maximisation distribution over actions. Under mild conditions, CQL produces a lower bound: $\hat{Q}^{\pi_\beta}(s, a) \leq Q^{\pi_\beta}(s, a)$ for all $(s, a) \in \mathcal{D}$.

TD3+BC (offline). Simple offline RL baseline (Fujimoto & Gu, 2021): $\pi_\phi = \arg\max_\pi \mathbb{E}_{(s,a) \sim \mathcal{D}}\left[\lambda Q_\phi(s, \pi(s)) - \right.$ The BC term $(\pi(s) - a)^2$ regularises the policy to stay close to the dataset, preventing OOD actions. Despite its simplicity, TD3+BC is competitive with complex methods like CQL.

100.3 RLHF and Constitutional AI: Mathematics of Alignment

Preference modelling. Given pairs of outputs (y_1, y_2) for prompt x, human annotators provide preference labels $y_w \succ y_l$. The Bradley-Terry model assumes:

$$P(y_1 \succ y_2 \mid x) = \sigma(r(x, y_1) - r(x, y_2)), \quad \sigma(z) = 1/(1 + e^{-z})$$

This is equivalent to logistic regression on preference differences. The reward model $r_\phi(x, y)$ is a neural network (typically the base LM + a scalar head). Training loss:

$$\mathcal{L}_{\text{BT}}(\phi) = -\mathbb{E}_{(x,y_w,y_l)}\left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right]$$

Plackett-Luce ranking. Generalise Bradley-Terry to full rankings over K outputs:

$$P(y_{\sigma(1)} \succ \cdots \succ y_{\sigma(K)} \mid x) = \prod_{k=1}^{K} \frac{\exp(r(x, y_{\sigma(k)}))}{\sum_{j=k}^{K} \exp(r(x, y_{\sigma(j)}))}$$

This is used when annotators provide a full ranking rather than pairwise comparisons.

RLHF objective.

$$\max_\theta \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(\cdot|x)}\left[r_\phi(x, y)\right] - \beta\, D_{\text{KL}}(\pi_\theta(\cdot|x) \,\|\, \pi_{\text{ref}}(\cdot|x))$$

The optimal closed-form solution is $\pi^*(y|x) = \pi_{\text{ref}}(y|x)\exp(r_\phi(x, y)/\beta)/Z(x)$.

DPO derivation. Substituting $r_\phi = \beta \log(\pi_\theta/\pi_{\text{ref}}) + \beta \log Z(x)$ into the Bradley-Terry loss and noting that $Z(x)$ cancels in the difference:

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)\right]$$

This is a purely supervised loss: no RL loop, no reward model inference at training time.

Constitutional AI (Bai et al., 2022). Two-phase approach: Phase 1 (SL-CAI): generate critiques and revisions of harmful outputs using a fixed set of principles; train on revised outputs (supervised). Phase 2 (RL-CAI): use the constitutionalised SL model as preference annotator (AI Feedback); train with RLHF on AI-generated preferences. This allows scalable oversight: the AI's own preferences (guided by principles) replace human labelling.

KL-regularised RLHF: token-level MDP. In LLM RLHF, actions are tokens: $a_t \in \mathcal{V}$ (vocabulary). State is the prompt + generated prefix. The reward $r_\phi(x, y)$ is typically given only at the final token (episodic). The per-token KL penalty is applied at each step: $r_t^{\text{token}} = -\beta \log(\pi_\theta(a_t|s_t)/\pi_{\text{ref}}(a_t|s_t))$ for $t < T$ and $r_T = r_\phi(x, y) - \beta \log(\pi_\theta(a_T|s_T)/\pi_{\text{ref}}(a_T|s_T))$. The value function $V_\phi(s_t) = \mathbb{E}[G_t|s_t]$ is trained alongside the policy.

PART 101

Hierarchical Reinforcement Learning

Hierarchical RL decomposes long-horizon tasks into sub-goals and sub-policies. We formalise options, semi-MDPs, and the options framework, then derive the multi-timescale Bellman equations and goal-conditioned policy learning.

101.1 The Options Framework

Motivation. Long-horizon RL with sparse rewards is extremely hard: the agent must discover a chain of correct actions stretching over hundreds of timesteps before seeing any signal. Hierarchical RL addresses this by learning reusable temporally-extended actions (*options*).

Option

An *option* $o = (\mathcal{I}_o, \pi_o, \beta_o)$ consists of:

- $\mathcal{I}_o \subseteq \mathcal{S}$: the *initiation set* (states where o can be invoked);
- $\pi_o : \mathcal{S} \times \mathcal{A} \to [0,1]$: the *intra-option policy*;
- $\beta_o : \mathcal{S} \to [0,1]$: the *termination function* ($\beta_o(s)$ is the probability of terminating in s).

Semi-Markov Decision Processes. Executing an option o from state s generates a trajectory of variable length k (the option duration) before terminating. This induces a *semi-MDP* over options: the transition kernel $\mathcal{P}_o(s' \mid s)$ and reward $\mathcal{R}_o(s)$ are computed by marginalising over the option trajectory.

Options Bellman equation. Let $Q_\Omega(s, o)$ be the value of executing option o from s under the option-policy μ:

$$Q_\Omega(s,o) = \sum_{s',k} p(s', k \mid s, o) \left[\sum_{t=0}^{k-1} \gamma^t r_t + \gamma^k V_\Omega(s') \right]$$

where $V_\Omega(s) = \max_o Q_\Omega(s,o)$ and $p(s', k|s, o)$ is the distribution over exit states and durations.

Intra-option Q-learning. Update the option value on every primitive step t:

$$Q_\Omega(s_t, o_t) \leftarrow Q_\Omega(s_t, o_t) + \alpha \left[r_t + \gamma U_\Omega(s_{t+1}, o_t) - Q_\Omega(s_t, o_t) \right]$$

where the *option continuation value* is:

$$U_\Omega(s', o) = (1 - \beta_o(s')) Q_\Omega(s', o) + \beta_o(s') V_\Omega(s')$$

This update uses every primitive transition, not just option terminations, greatly improving sample efficiency.

101.2 Goal-Conditioned Policies and HER

Goal-conditioned policies. Augment the state with a goal $g \in \mathcal{G}$: $\pi(a \mid s, g)$. A universal value function $Q(s, a, g)$ generalises across goals. The reward is typically $r(s, a, g) = -\mathbf{1}[d(s', g) > \varepsilon]$ (sparse: -1 unless at goal).

Hindsight Experience Replay (HER). (Andrychowicz et al., 2017). After failing to reach goal g, reuse the trajectory with the actual final state s_T as a "virtual goal": for the same trajectory $(s_0, a_0, \ldots, s_T)$, store transitions with goal $g' = s_T$, which generates reward $r = 0$ at termination. This converts failed episodes into informative training signal.

HER samples goals from the replay buffer using a strategy $g \sim \mathcal{G}$:

- **final**: $g' = s_T$ (final state of episode);
- **future**: $g' = s_{t'}$ for $t' > t$ (future state in same episode);
- **random**: g' uniform from replay buffer.

The "future" strategy performs best empirically. HER enables learning in environments where random exploration almost never reaches the goal.

Example — HER on a grid maze

Goal: reach position $(10, 10)$ in a 12×12 grid. Random agent reaches $(10, 10)$ with probability $\sim 1/144$ per episode — essentially never. With HER (future strategy): every episode contributes $\sim k$ virtual goal transitions where k is episode length. After 100 episodes: ~ 5000 virtual transitions covering most of the grid. The Q-function learns the value landscape from all virtual goals simultaneously, enabling rapid generalization to the true goal.

101.3 Feudal Networks and Subgoal Discovery

Feudal architecture (Vezhnevets et al., 2017). Two-level hierarchy:

- **Manager**: operates at timescale c (every c steps); outputs subgoal $g_t \in \mathbb{R}^d$ in latent space.
- **Worker**: operates at every step; receives (s_t, g_t) and maximises $r_t^W = \frac{1}{c}\sum_{i=0}^{c-1} \cos(s_{t+i+1} - s_t, g_t)$ (cosine similarity of state change with subgoal direction).

Manager gradient (DILATED LSTM, dilation c):

$$\nabla_\phi J(\phi) = \mathbb{E}\left[\sum_t A_t^M \nabla_\phi \log \pi_\phi^M(g_t \mid s_t)\right]$$

where $A_t^M = \sum_{i=0}^{c-1} \gamma^i r_{t+i}^{extrinsic} - V^M(s_t)$.

Automatic subgoal discovery. Options with learned initiation and termination functions: DIAYN (Eysenbach et al., 2018) learns diverse skills by maximising mutual information between skills z and states:

$$J(\theta, q) = \mathbb{E}_{z\sim p(z), s\sim\pi_z}\left[\log q(z \mid s) - \log p(z)\right] + H[\pi_z]$$

where $q(z \mid s)$ is a discriminator network trained to identify which skill generated state s. This creates a repertoire of distinguishable, diverse behaviours without any extrinsic reward.

Part 102

Meta-Reinforcement Learning

Meta-RL learns a learning algorithm itself: after adapting on a few episodes in a new task, the agent achieves near-optimal performance. We cover MAML, RL^2, and the mathematical connections to Bayes-optimal RL and in-context learning.

102.1 The Meta-RL Problem

Task distribution. Meta-RL assumes a distribution $p(\mathcal{M})$ over MDPs $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}_\mathcal{M}, \mathcal{R}_\mathcal{M}, \gamma)$, all sharing the same state/action spaces but differing in dynamics or rewards. At meta-test time, a new task $\mathcal{M}_{\text{test}} \sim p(\mathcal{M})$ is drawn; the agent has K *adaptation episodes* to learn task-specific behaviour, then is evaluated.

The meta-objective: maximise expected performance after K adaptation episodes:

$$J_{\text{meta}}(\theta) = \mathbb{E}_{\mathcal{M}\sim p(\mathcal{M})}\left[J_\mathcal{M}(\pi_{\theta'})\right]$$

where $\theta' = \text{Adapt}(\theta, \mathcal{M})$ uses K episodes from $\mathcal{M}$.

The meta-learning problem. Standard RL learns a single policy π_θ for a fixed MDP $\mathcal{M}$. Meta-RL considers a *distribution of tasks* $p(\mathcal{M})$ and aims to learn a *learning algorithm* (or initial parameters) that enables rapid adaptation on new tasks drawn from $p(\mathcal{M})$.

Meta-RL Objective

Let $\mathcal{L}_{\mathcal{M}}(\theta)$ be the performance on task $\mathcal{M}$ after K gradient steps (or K rollouts) of adaptation starting from θ. Meta-RL optimises:

$$\theta^* = \arg\max_\theta \mathbb{E}_{\mathcal{M}\sim p(\mathcal{M})}[\mathcal{L}_{\mathcal{M}}(\mathcal{U}_K(\theta, \mathcal{D}^{\text{train}}_{\mathcal{M}}))]$$

where $\mathcal{U}_K$ is a K-step adaptation operator (gradient descent, LSTM update, etc.) and $\mathcal{D}^{\text{train}}_{\mathcal{M}}$ is the small task-specific training dataset.

Few-shot adaptation. At meta-test time, the agent is given K episodes on a new task $\mathcal{M}_{\text{test}}$ and must perform well by the $(K+1)$-th episode. This is in contrast to standard RL, which requires thousands of episodes per task.

Task structure. Meta-RL is meaningful only when tasks share structure (similar dynamics, different reward functions; or similar reward structure, slightly different transitions). Examples:

- Locomotion tasks with different goal directions (same robot dynamics, different targets).
- Multi-armed bandits with different reward distributions.
- Navigation in mazes with different goal locations.

Approaches. Three main paradigms:

1. **Gradient-based** (MAML): find θ such that a few gradient steps give good performance.
2. **Recurrence-based** (RL^2): meta-policy is an RNN that "remembers" intra-task experience.
3. **Context-based**: encode task identity into a latent z and condition policy on z.

102.2 MAML for RL

Model-Agnostic Meta-Learning (MAML) (Finn et al., 2017). Adaptation via gradient descent:

$$\theta'_{\mathcal{M}} = \theta - \alpha_{\text{inner}} \nabla_\theta J_{\mathcal{M}}(\theta)$$

Meta-update (outer loop):

$$\theta \leftarrow \theta + \alpha_{\text{outer}} \nabla_\theta \sum_{\mathcal{M}} J_{\mathcal{M}}(\theta'_{\mathcal{M}})$$

The outer gradient requires differentiating through the inner gradient step — a second-order derivative.

First-order MAML (FOMAML). Approximate by dropping second-order terms: $\nabla_\theta J(\theta') \approx \nabla_{\theta'} J(\theta')$. Empirically competitive with full MAML, much cheaper ($O(d)$ instead of $O(d^2)$ memory).

MAML for RL: ProMP. The inner loop uses TRPO/PPO; the outer loop uses the performance difference lemma. With $K = 1$ adaptation episode per task and N outer-loop tasks:

$$\mathcal{L}_{\text{meta}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{\text{CLIP}}(\theta_i')$$

The meta-gradient: $\nabla_\theta \mathcal{L}_{\text{meta}} = \frac{1}{N} \sum_i \nabla_\theta \mathcal{L}_{\text{CLIP}}(\theta - \alpha \nabla_\theta \mathcal{L}_i^{\text{adapt}})$. This requires backpropagating through the policy gradient step — computable via Hessian-vector products.

102.3 RL2 and In-Context RL

RL^2 (Duan et al., 2016). Treat the K-episode adaptation phase as a single long episode for an RNN (LSTM). At each step, the RNN receives $(s_t, a_{t-1}, r_{t-1}, d_{t-1})$ where d is a done flag. The LSTM hidden state h_t encodes all information about the task; the output is a policy $\pi_\theta(a_t \mid h_t)$. The RNN is trained with standard RL (PPO) on the meta-objective across tasks. The optimal RNN implements the Bayes-optimal policy for the task distribution — it is doing implicit posterior inference over tasks in its hidden state.

Bayes-optimality. The Bayes-optimal policy:

$$\pi^*_{BO}(a \mid h_t) = \arg\max_a \mathbb{E}_{\mathcal{M} \sim p(\mathcal{M} \mid h_t)} \left[Q^*_{\mathcal{M}}(s_t, a) \right]$$

maintains a belief $b(\mathcal{M}) = p(\mathcal{M} \mid h_t)$ over tasks (posterior given history h_t), and acts to maximise expected value under this belief. Computing this exactly requires solving a POMDP over the space of tasks. RL^2 and similar methods implicitly approximate π^*_{BO} via gradient descent.

Theorem: RL^2 convergence to Bayes-optimal

If the RNN has sufficient capacity and is trained to convergence on the meta-RL objective, the resulting policy approaches the Bayes-optimal policy for the task distribution $p(\mathcal{M})$.

This follows because the meta-RL objective $\mathbb{E}_{\mathcal{M}}[J_{\mathcal{M}}(\pi)]$ is maximised by π^*_{BO}, and universal approximation by RNNs ensures such policies are representable.

Example — RL^2 on multi-armed bandits

Task: K-armed bandit, rewards $\mu_i \sim \text{Uniform}(0, 1)$, new means each episode. $T = 100$ steps. The Bayes-optimal policy is Thompson Sampling. RL^2 learns to: (a) explore in the first 20 steps, (b) commit to the empirical best arm. The LSTM hidden state encodes the empirical success rates $\hat{\mu}_i$ and trial counts N_i. After meta-training, RL^2 achieves regret within 10% of Thompson Sampling on held-out bandit instances.

Part 103

Reward Shaping and Intrinsic Motivation

Sparse rewards make RL extremely slow. We formalise potential-based reward shaping, derive the invariance theorem, and survey intrinsic motivation methods: curiosity, empowerment, and information gain.

103.1 Potential-Based Reward Shaping

The shaping problem. Suppose the agent receives reward $r(s, a, s')$. Adding a shaping term $F(s, a, s')$ creates a new MDP with reward $r + F$. If F is arbitrary, the optimal policy can change — potentially dramatically. We want shaping that preserves the optimal policy.

Potential-Based Shaping (Ng et al., 1999)

A shaping reward $F(s, a, s') = \gamma\Phi(s') - \Phi(s)$ for any *potential function* $\Phi : \mathcal{S} \to \mathbb{R}$ preserves the optimal policy: π^* in the shaped MDP equals π^* in the original.

Proof. The shaped Q-function:

$$Q_F^*(s, a) = Q^*(s, a) + \Phi(s) \cdot \text{const}$$

More precisely: define $Q_F(s, a) = Q(s, a) + \Phi(s)$. Then:

$$\begin{aligned} Q_F(s, a) &= (r + \gamma\Phi(s') - \Phi(s)) + \gamma \max_{a'} Q_F(s', a') - \gamma\Phi(s') \\ &= r + \gamma \max_{a'} Q(s', a') = Q(s, a) \end{aligned}$$

So $\arg\max_a Q_F(s, a) = \arg\max_a Q(s, a)$. Q.E.D.

Good potential functions. Popular choices:

- $\Phi(s) = V^*(s)$ (ideal but unknown): shapes to $r_F = 0$ everywhere — pure intrinsic.
- $\Phi(s) = \hat{V}^\pi(s)$ (learned critic): guides agent toward states of high value.
- $\Phi(s) = -d(s, g)$ (negative distance to goal): encourages approach.
- $\Phi(s) = \log \rho^{1/2}(s)$ (sub-goal density): reward for reaching sub-goals.

The shaping theorem. Reward shaping adds a potential-based bonus $F(s, a, s') = \gamma\Phi(s') - \Phi(s)$ to the environment reward: $\tilde{r}(s, a, s') = r(s, a, s') + F(s, a, s')$. Ng, Russel & Harada (1999) proved:

Potential-Based Shaping Theorem

For any potential function $\Phi : \mathcal{S} \to \mathbb{R}$, the shaped reward $\tilde{r} = r + \gamma\Phi(s') - \Phi(s)$ satisfies: every optimal policy π^* under $\tilde{r}$ is also optimal under r, and vice versa.

Proof. The Q-functions are related by $\tilde{Q}^\pi(s, a) = Q^\pi(s, a) - \Phi(s)$, so $\arg\max_a \tilde{Q}^\pi(s, a) = \arg\max_a Q^\pi(s, a)$. The optimal policy is unchanged. □

Choosing Φ. Good potential functions encode domain knowledge:

- Navigation: $\Phi(s) = -d(s, s_{\text{goal}})$ (negative distance to goal).
- Robotics: $\Phi(s) =$ progress along a reference trajectory.
- RLHF: $\Phi(s) =$ language model log-probability of current partial generation.

Non-potential shaping. Arbitrary shaped rewards $\tilde{r} = r + F$ (without the potential structure) can change the set of optimal policies. For example, a constant bonus $F = c$ is potential-based ($\Phi \equiv 0$) and safe; but $F(s, a) = 1[\text{new state visited}]$ (exploration bonus) is *not* potential-based and alters optimal behaviour (Dewey, 2014). This is used intentionally in exploration but must be applied carefully.

103.2 Intrinsic Motivation

Empowerment (Klyubin et al., 2005). The empowerment of state s is the channel capacity of the mapping from n-step action sequences $a_{t:t+n}$ to future states s_{t+n}:

$$\mathcal{E}(s) = \max_{p(a_{t:t+n})} I(a_{t:t+n}; s_{t+n} \mid s_t = s)$$

I is mutual information. High empowerment means the agent has many ways to influence its future — a proxy for agency and capability. Empowerment maximisation yields intrinsically motivated exploration of states with many reachable futures, even without extrinsic rewards.

Information-gain exploration. In a Bayesian model with posterior $p(\mathcal{M} \mid h_t)$:

$$r_t^+ = I(\mathcal{M}; s_{t+1} \mid h_t, a_t) = H(s_{t+1} \mid h_t, a_t) - \mathbb{E}_{\mathcal{M}}[H(s_{t+1} \mid h_t, a_t, \mathcal{M})]$$

Information gain is high for actions that maximally reduce uncertainty about $\mathcal{M}$. This is the formally optimal exploration strategy for Bayes-optimal RL.

VIME (Variational Information Maximising Exploration) (Houthooft et al., 2016). Approximate the information gain using a variational Bayes neural network model $q(\theta|D)$:

$$r_t^+ \approx D_{\mathrm{KL}}(q(\theta|D_t) \| q(\theta|D_{t-1}))$$

The bonus is the KL divergence of the model posterior before and after observing (s_t, a_t, s_{t+1}). This rewards transitions that change the model's beliefs — genuine novelty rather than mere visitation count.

Motivation. In sparse reward environments, an agent may never receive a reward signal during random exploration. Intrinsic motivation provides an internal reward signal based on novelty, surprise, or prediction error, encouraging exploration independent of the extrinsic reward.

Count-based exploration. In tabular MDPs, the visit count $N(s)$ directly measures novelty. The pseudo-count bonus:

$$r^i(s) = \beta / \sqrt{N(s)}$$

gives higher reward to rarely visited states. For function approximation, pseudo-counts are estimated from a density model $\hat{\rho}(s)$ via: $\hat{N}(s) \approx \hat{\rho}(s) \cdot n/(1 - \hat{\rho}(s))$.

Prediction-error motivation (ICM). The Intrinsic Curiosity Module (Pathak et al., 2017) trains:

- A feature encoder $\phi : \mathcal{S} \to \mathbb{R}^d$.
- An inverse model: predict a_t from $(\phi(s_t), \phi(s_{t+1}))$ to learn action-relevant features.
- A forward model: predict $\hat{\phi}(s_{t+1})$ from $(\phi(s_t), a_t)$.

Intrinsic reward: $r_t^i = \frac{\eta}{2}\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|^2$, i.e., forward-model prediction error.

Random Network Distillation (RND). A simpler alternative: fix a random target network $f_{\text{tgt}} : \mathcal{S} \to \mathbb{R}^k$ and train a predictor f_{pred} to match it. Prediction error:

$$r^i(s) = \|f_{\text{pred}}(s) - f_{\text{tgt}}(s)\|^2$$

is high for novel states (the predictor hasn't seen them) and low for familiar states. RND is computationally cheap, requires no inverse model, and scales to Atari-level environments.

Example — Montezuma's Revenge

This Atari game is the canonical hard-exploration benchmark: the reward is extremely sparse and requires long sequences of specific actions. Model-free RL with no exploration bonus scores near 0. With RND bonus, agents reach $> 10{,}000$ points, discovering hundreds of rooms, nearly matching human performance.

103.3 Curriculum Learning in RL

Automatic curriculum generation. Present tasks in order of increasing difficulty: start with easy goals (reachable with random policy), gradually shift toward harder targets. The learning progress (LP) criterion: select tasks with highest improvement rate:

$$g^* = \arg\max_g \frac{\Delta J(g)}{\Delta t} = \arg\max_g \frac{J_t(g) - J_{t-\tau}(g)}{\tau}$$

Goal GAN (Florensa et al., 2018). Maintain a GAN that generates goals of appropriate difficulty. Discriminator labels goals as "feasible" if they are achievable with probability in $[p_{\min}, p_{\max}]$. Generator learns to produce goals at the boundary of current competence (GOID region: Goals of Intermediate Difficulty).

PLR (Prioritised Level Replay) (Jiang et al., 2021). For procedurally generated environments: score levels by their "learning potential" (staleness × TD error magnitude):

$$P(l) \propto e^{\beta \cdot \text{score}(l)} = e^{\beta \cdot (c_1 \cdot \text{staleness}(l) + c_2 \cdot \text{error}(l))}$$

Replay the most instructive levels frequently; explore new levels for diversity.

Part 104

Inverse Reinforcement Learning and Imitation Learning

Given expert trajectories, IRL recovers the reward function that explains the behaviour. We derive maximum entropy IRL, maximum causal entropy, and the GAIL framework, with full mathematical treatment of the inverse problem.

104.1 The Inverse Reinforcement Learning Problem

Forward vs. inverse RL. Forward RL: given reward $\mathcal{R}$, find optimal policy π^*. Inverse RL (IRL): given expert trajectories $\{\tau_i^E\}$, recover reward $\mathcal{R}^*$ such that $\pi^*_{\mathcal{R}^*} \approx \pi^E$ (the expert).

Ambiguity. IRL is ill-posed: any constant $\mathcal{R}$ makes all policies optimal; $\mathcal{R}$ and $c \cdot \mathcal{R}$ (scaling) yield the same optimal policy. Regularisation and maximum entropy priors resolve this.

IRL as optimisation

Given expert feature expectations $\mu_E = \mathbb{E}_{\pi^E}[\phi(s,a)]$ and a parameterised reward $R_w(s,a) = w^\top \phi(s,a)$, find w such that:

$$\mu_E = \mu_{\pi^*(w)} = \mathbb{E}_{\pi^*(w)}[\phi(s,a)]$$

This is feasible matching: the expert's feature expectations are matched by the optimal policy under R_w.

Inverse reinforcement learning (IRL). In imitation learning, we observe expert demonstrations $\mathcal{D} = \{(s_t^{(i)}, a_t^{(i)})\}$ and want to recover a reward function r such that the expert is (approximately) optimal under r.

IRL Problem

Given: MDP$\backslash r$ (MDP without reward), expert policy π^E (or demonstrations from it).
Find: reward r^* such that $\pi^E = \arg\max_\pi V_r^\pi$ (or approximately so).

Ambiguity. IRL is severely ill-posed: the reward $r \equiv 0$ makes *any* policy optimal. Constraints are needed:

- Maximum margin: find r with largest margin between expert and all other policies.
- Maximum entropy: find r such that the induced distribution over trajectories has maximum entropy subject to matching expert feature expectations.
- Bayesian IRL (Ramachandran & Amir, 2007): posterior $P(r|\mathcal{D}) \propto P(\mathcal{D}|r)P(r)$.

Feature-based reward. Assume $r(s,a) = \theta^\top \phi(s,a)$ where ϕ are hand-crafted features. IRL recovers θ. The expert's feature expectations: $\mu^E = \mathbb{E}_{\pi^E}[\sum_t \gamma^t \phi(s_t, a_t)]$. The IRL constraint: $\theta^{*\top}\mu^E \geq \theta^{*\top}\mu^\pi$ for all π, i.e., the expert has higher return under r^* than any other policy.

Limitations. IRL requires solving the forward RL problem (computing π_r^*) in an inner loop, which is computationally expensive. Apprenticeship learning and MaxEnt IRL address this via efficient feature matching without solving the full RL problem at each step.

104.2 Maximum Entropy IRL

Maximum entropy IRL (Ziebart et al., 2008). Instead of feasibility matching, adopt the *maximum entropy principle*: among all distributions consistent with the expert constraints, choose the one with maximum entropy.

The IRL likelihood is:

$$P(\tau \mid R_w) = \frac{1}{Z(w)} \exp\left(\sum_t R_w(s_t, a_t)\right), \quad Z(w) = \sum_\tau \exp(R(\tau))$$

Maximise log-likelihood over expert demonstrations:

$$\max_w \sum_i \log P(\tau_i^E \mid R_w) = \sum_i R_w(\tau_i^E) - N \log Z(w)$$

Gradient. $\nabla_w \log Z(w) = \mathbb{E}_{\pi^*(w)}[\phi(s,a)] = \mu_{\pi^*(w)}$. So the gradient of log-likelihood is: $\nabla_w \mathcal{L} = \mu_E - \mu_{\pi^*(w)}$. This has a beautiful interpretation: update w to reduce the feature expectation gap between expert and policy. The inner loop (forward RL to compute $\mu_{\pi^*(w)}$) is expensive; approximate methods use sampling.

Maximum causal entropy IRL (Ziebart, 2010). The causal entropy $H_C(\pi) = -\mathbb{E}_\pi[\sum_t \log \pi(a_t \mid s_t)]$ accounts for the agent's causal structure (actions cause future states). The soft Q-function:

$$Q^{\text{soft}}(s,a;w) = R_w(s,a) + \gamma \,\text{softmax}_{a'} Q^{\text{soft}}(s',a';w)$$

where $\text{softmax}_a f(a) = \log \sum_a \exp(f(a))$. The stochastic policy $\pi(a \mid s) = \exp(Q^{\text{soft}}(s,a) - V^{\text{soft}}(s))$ maximises causal entropy subject to matching expected reward — a principled probabilistic expert model.

MaxEnt IRL objective. Ziebart et al. (2008) proposed finding the reward $r_\theta = \theta^\top \phi$ that maximises the likelihood of the demonstrations under the maximum entropy principle:

$$\max_\theta \ \mathbb{E}_{\tau\sim\mathcal{D}}[\log P_\theta(\tau)], \qquad P_\theta(\tau) = \frac{1}{Z(\theta)} \exp\left(\sum_t r_\theta(s_t, a_t)\right)$$

where $Z(\theta) = \sum_\tau \exp(\sum_t r_\theta(s_t, a_t))$ is the partition function over all trajectories.

Gradient. The gradient of the log-likelihood is the difference between expert and model feature expectations:

$$\nabla_\theta \mathcal{L} = \mu^E - \mathbb{E}_{\tau\sim P_\theta}[\phi(\tau)] = \underbrace{\mathbb{E}_{\tau\sim\mathcal{D}}[\phi(\tau)]}_{\text{expert}} - \underbrace{\sum_\tau P_\theta(\tau)\phi(\tau)}_{\text{model}}$$

Gradient ascent moves θ to increase expert feature expectations relative to model expectations.

Computing the model expectation. The sum over all trajectories requires computing: $\mu_\theta(s) = \sum_{t,\tau} P_\theta(\tau)\phi(s_t)$, which is done by a two-pass algorithm:

1. **Backward pass**: compute soft values $Q_{\text{soft}}(s, a)$ and $Z(s)$ via backward DP.
2. **Forward pass**: compute state visitation frequencies $D(s)$ via forward DP on the soft policy.

The model feature expectation is $\mu_\theta = \sum_s D(s)\phi(s)$.

GAIL (Ho & Ermon, 2016). Generalises MaxEnt IRL to the GAN framework: the discriminator distinguishes expert from agent trajectories; the generator (agent policy) tries to fool the discriminator. GAIL is reward-free and scales to high-dimensional continuous actions.

104.3 Behavioural Cloning vs. IRL

Behavioural cloning (BC). Supervised learning directly on (s_t, a_t) pairs from expert: $\pi_\theta = \arg\min_\theta \mathbb{E}_{(s,a)\sim D_E}[\mathcal{L}(\pi_\theta(s), a)]$. Failure mode: *compounding errors*. Small mistakes in early states lead to states not covered by the expert dataset; the policy has no guidance and errors accumulate: if ε_t is per-step error, the total error at step T scales as $O(\varepsilon T^2)$.

DAgger (Dataset Aggregation) (Ross et al., 2011). Iterative correction:

1. Train π_i via BC on current dataset $\mathcal{D}_i$.
2. Roll out π_i to collect states $\mathcal{D}'_i = \{s_t\}$.
3. Query expert for actions: $\mathcal{D}''_i = \{(s_t, \pi^E(s_t))\}$.
4. $\mathcal{D}_{i+1} \leftarrow \mathcal{D}_i \cup \mathcal{D}''_i$.

DAgger reduces compounding errors to $O(\varepsilon T)$ by training on the agent's own visited states.

Example — Compounding error illustration

Expert demonstrates driving along a road. BC trains to mimic. At test time, a small perturbation puts the car slightly off-centre (state not in training data). BC has no training signal for this state and produces a random action; the car drifts further off-road,

into an even more out-of-distribution state. After $T = 10$ steps, the error has drifted $O(T^2\varepsilon) = O(100\varepsilon)$ — catastrophic. DAgger: query expert at the perturbed state; the dataset now includes off-centred states and recovery actions. After 5 iterations: error is $O(T\varepsilon) = O(10\varepsilon)$ — 10× improvement.

Behavioural Cloning (BC). The simplest imitation approach: treat the demonstrations as a supervised dataset and minimise the cross-entropy loss:

$$\mathcal{L}_{\text{BC}}(\theta) = -\mathbb{E}_{(s,a)\sim\mathcal{D}}[\log \pi_\theta(a|s)]$$

Covariate shift problem. During training, (s, a) pairs come from the expert's state distribution d^E. During test, the policy π_θ induces its own state distribution $d^{\pi_\theta} \neq d^E$. Small errors compound: if the agent makes an ϵ mistake per step, after T steps the state distribution diverges as $O(\epsilon T^2)$ (Ross & Bagnell, 2010).

DAgger (Ross et al., 2011). Dataset Aggregation corrects covariate shift:

1. Collect trajectories from current policy π_θ (not the expert).
2. Query the expert to label the states visited: $a^E = \pi^E(s)$ for each s encountered.
3. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s, a^E)\}$.
4. Retrain π_θ on $\mathcal{D}$.

DAgger achieves $O(\epsilon T)$ regret (linear rather than quadratic), matching the optimal imitation bound.

BC vs IRL comparison.

- **BC**: no reward needed, simple, fast. Fails under covariate shift.
- **IRL**: recovers reward, generalises to new environments, expensive (inner RL loop).
- **GAIL**: intermediate — uses discriminator as implicit reward, scalable, no explicit reward.

For high-quality dense demonstrations, BC with DAgger often suffices. For sparse demonstrations or when generalisation to new dynamics is needed, IRL-based methods are preferred.

PART 105

Safe Reinforcement Learning and Constrained MDPs

Safety in RL requires policies that respect constraints during both training and deployment. We formalise Constrained MDPs, derive the Lagrangian approach, and cover safe policy improvement bounds and worst-case RL.

105.1 Constrained Markov Decision Processes

Constrained MDP (CMDP)

A *Constrained MDP* is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \{\mathcal{C}_k\}_{k=1}^m, \{d_k\}_{k=1}^m, \gamma)$ where $\mathcal{C}_k(s, a)$ is the

k-th cost function and d_k is its budget. The optimisation problem:

$$\max_\pi J(\pi) = \mathbb{E}_\pi\left[\sum_t \gamma^t r_t\right] \quad \text{s.t.} \quad J^{C_k}(\pi) = \mathbb{E}_\pi\left[\sum_t \gamma^t C_k(s_t, a_t)\right] \le d_k \quad \forall k$$

Lagrangian relaxation. Form the Lagrangian:

$$\mathcal{L}(\pi, \lambda) = J(\pi) - \sum_k \lambda_k (J^{C_k}(\pi) - d_k)$$

Under Slater's condition (there exists a feasible policy with strict inequality constraints), strong duality holds: $\max_\pi \min_{\lambda \ge 0} \mathcal{L}(\pi, \lambda) = \min_{\lambda \ge 0} \max_\pi \mathcal{L}(\pi, \lambda)$. The dual problem: find λ^* such that $\pi^*_{\lambda^*}$ (optimal policy for $\mathcal{R} - \sum_k \lambda_k C_k$) is feasible.

Primal-dual gradient updates. Alternate:

$$\pi \leftarrow \pi + \alpha_\pi \nabla_\pi \mathcal{L}(\pi, \lambda), \qquad \lambda_k \leftarrow [\lambda_k + \alpha_\lambda (J^{C_k}(\pi) - d_k)]_+$$

The $[\cdot]_+$ ensures $\lambda \ge 0$. This converges to a saddle point (π^*, λ^*) under appropriate step sizes.

Constrained MDPs (CMDPs). In many applications, the agent must satisfy safety constraints in addition to maximising reward. A CMDP augments the MDP with a set of constraint cost functions $c_k : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, $k = 1, \dots, K$, and threshold values d_k:

$$\max_\pi J_r(\pi) \quad \text{subject to} \quad J_{c_k}(\pi) \le d_k \quad \forall k$$

where $J_{c_k}(\pi) = \mathbb{E}_\pi[\sum_{t=0}^\infty \gamma^t c_k(s_t, a_t)]$ is the expected cumulative cost under policy π.

Lagrangian relaxation. The standard approach dualises the constraints:

$$\max_\pi \min_{\lambda_k \ge 0} \; J_r(\pi) - \sum_k \lambda_k (J_{c_k}(\pi) - d_k)$$

Under strong duality (holds for CMDPs with linear-program structure), the min-max equals the constrained maximum. The dual variables λ_k are the *constraint prices*: $\lambda_k > 0$ whenever constraint k is active.

Primal-Dual Policy Gradient for CMDPs

Alternate between:

- Policy update (gradient ascent on Lagrangian): $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \mathcal{L}(\theta, \lambda)$.
- Dual update (gradient ascent on constraint violation): $\lambda_k \leftarrow [\lambda_k + \alpha_\lambda (J_{c_k}(\pi_\theta) - d_k)]_+$.

Occupancy measure LP. CMDPs can be solved exactly (tabular case) as a linear program over the occupancy measure $\rho(s, a) = \sum_t \gamma^t P(s_t = s, a_t = a | \pi)$:

$$\max_{\rho \ge 0} \sum_{s,a} r(s,a)\rho(s,a) \quad \text{s.t.} \quad \sum_{s,a} c_k(s,a)\rho(s,a) \le d_k, \; \text{flow constraints}$$

This gives the globally optimal constrained policy without alternating gradient updates.

105.2 CPO and Safe Policy Improvement

Constrained Policy Optimisation (CPO) (Achiam et al., 2017). Analogue of TRPO for CMDPs. The update:

$$\pi_{k+1} = \arg\max_{\pi} L^{\pi_k}(\pi) \quad \text{s.t.} \quad J^{C_j}(\pi_{k+1}) \le d_j + \varepsilon_j \quad \forall j, \quad \overline{D}_{KL}(\pi_k, \pi_{k+1}) \le \delta$$

where $L^{\pi_k}(\pi)$ is the surrogate objective and ε_j is a small slack.

Safety guarantee. CPO provides a monotonic safety improvement theorem: if $J^{C_j}(\pi_k) \le d_j$, then $J^{C_j}(\pi_{k+1}) \le d_j + O(\delta/(1-\gamma))$ (approximately feasible). Exact feasibility can be guaranteed by projecting back to the constraint set if violated.

Worst-case RL. Robustness to environment perturbations:

$$\pi^* = \arg\max_{\pi} \min_{\tilde{\mathcal{P}} \in \mathcal{U}} J(\pi; \tilde{\mathcal{P}})$$

where $\mathcal{U} = \{\tilde{\mathcal{P}} : d(\tilde{\mathcal{P}}, \mathcal{P}) \le \epsilon\}$ is an uncertainty set. Robust MDPs can be solved by modified value iteration: $V^*(s) = \max_a \min_{\tilde{\mathcal{P}} \in \mathcal{U}_a} [\mathcal{R}(s,a) + \gamma \sum_{s'} \tilde{\mathcal{P}}(s'|s,a) V^*(s')]$. For rectangular uncertainty sets ($\mathcal{U}_a$ independent per (s,a)), this decomposes and is solvable in polynomial time.

Constrained Policy Optimisation (CPO). Achiam et al. (2017) extend the TRPO trust region to CMDPs. The CPO update solves:

$$\max_{\pi} \; J_r(\pi) \quad \text{s.t.} \quad J_{c_k}(\pi) \le d_k, \quad D_{\mathrm{KL}}(\pi_{\text{old}} \,\|\, \pi) \le \delta$$

Using a linear approximation to J_{c_k} and a quadratic approximation to the KL constraint (as in TRPO), CPO solves this constrained quadratic programme analytically at each step.

Safe policy improvement. Beyond CPO, the *safe policy improvement* framework (Laroche et al., 2019) provides probabilistic guarantees: given a baseline policy π_b and confidence $1-\delta$, find the best policy π such that $P(V^\pi \ge V^{\pi_b} - \varepsilon) \ge 1 - \delta$. This converts the constrained optimisation into a policy search with finite-sample safety certificates.

Lyapunov-based safety. An alternative to constraint-based methods uses a Lyapunov function $L : \mathcal{S} \to \mathbb{R}_{\ge 0}$ that certifies safety:

$$L(s_{t+1}) - L(s_t) \le -c\, L(s_t) \quad \Rightarrow \quad L(s_t) \le L(s_0) e^{-ct}$$

(exponential stability). The agent's policy must keep L non-increasing, constraining the action space at each state to the safe set $\{a : \mathbb{E}[L(s_{t+1}) | s_t, a] \le L(s_t)\}$.

Example — Safe RL in autonomous driving

A CPO agent on a lane-keeping task (Achiam et al., 2017) converges to near-optimal driving reward while keeping the cumulative constraint violation (crossings of lane boundaries) near zero. Unconstrained PPO achieves $\sim 10\%$ higher reward but with $5\times$ more constraint violations — illustrating the reward-safety Pareto trade-off in safety-critical RL.

105.3 Shield Synthesis and Formal Safety

Safety specifications. Formal safety requirements expressed as temporal logic formulas. *Linear Temporal Logic* (LTL) ϕ: $\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc\phi \mid \phi \mathcal{U} \phi$ (atomic, negation, conjunction, next, until). Safe RL objective: find π maximising $J(\pi)$ subject to $P(\pi \models \phi) = 1$ (or $\ge 1-\delta$).

Shield. A *shield* is a module $\mathcal{S}$ that intercepts unsafe actions: $a_{\text{safe}} = \mathcal{S}(s, a_{\text{agent}})$ — it overrides the agent whenever the proposed action would violate safety constraints. The shield is synthesised offline from the safety specification, independently of the RL agent. During training: the agent optimises within the safe action set; during deployment: the shield guarantees satisfaction of ϕ.

Example — Shield for a collision-avoidance task

Safety spec: "never enter the obstacle zone $\mathcal{D}$." Pre-compute safe actions from each state: $\mathcal{A}_{\text{safe}}(s) = \{a : P(s_{t+1} \in \mathcal{D} \mid s, a) = 0\}$. Agent trains PPO with action set restricted to $\mathcal{A}_{\text{safe}}(s_t)$ at each step. This guarantees zero collisions during training (a much stronger guarantee than penalising collisions in reward).

Shield synthesis. A *shield* is a runtime supervisor that intercepts the agent's action and overrides it if it would violate a safety specification. Shields are computed offline from a formal model and provide *hard guarantees* (unlike soft constraint methods).

Linear Temporal Logic (LTL) specifications. Safety properties are expressed in LTL:

- $\square p$: always p (invariant): e.g., $\square(\text{speed} < v_{\max})$.
- $\diamond p$: eventually p (reachability): e.g., $\diamond(\text{goal reached})$.
- $\square \diamond p$: always eventually p (liveness).
- $p \mathcal{U} q$: p until q.

Reactive synthesis. Given an LTL specification φ and a two-player game (agent vs. environment), reactive synthesis computes a *winning strategy* (if it exists): a policy that satisfies φ regardless of environment behaviour. The synthesis problem is PSPACE-complete for LTL and reduces to solving parity games over the Büchi automaton of φ.

Shield integration with RL. The shield $\mathcal{SH} : \mathcal{S} \times \mathcal{A} \to \mathcal{A}$ maps the agent's chosen action to a safe action:

$$a_t^{\text{safe}} = \mathcal{SH}(s_t, a_t^{\text{RL}}) = \begin{cases} a_t^{\text{RL}} & \text{if safe} \\ a^* \in \mathcal{A}_{\text{safe}}(s_t) & \text{otherwise (least restrictive safe action)} \end{cases}$$

The RL agent trains with the shielded environment; the shield's overrides are invisible to the reward model. This allows learning under safety constraints without constraining the policy gradient computation.

Part 106

RL for Large Language Models: RLHF, GRPO, and Reasoning

Modern LLM training uses RL at scale. We formalise the full pipeline: reward modelling from preferences, PPO for token-level MDPs, Group Relative Policy Optimisation, and the mathematics of chain-of-thought reasoning as a learned process reward.

106.1 The LLM-as-MDP Framework

Token-level MDP. A language model generates text token by token:

- State s_t: the concatenation of prompt x and generated prefix $(x, y_{1:t-1})$.
- Action $a_t = y_t$: the next token, $y_t \in \mathcal{V}$ (vocabulary of size $|\mathcal{V}| \sim 10^5$).

- Transition: deterministic, $s_{t+1} = (x, y_{1:t})$.
- Reward: typically $r_t = 0$ for $t < T$, $r_T = r_\phi(x, y_{1:T})$ (episodic).
- Policy: the LM itself, $\pi_\theta(y_t \mid x, y_{1:t-1})$.

Challenges specific to LLM-RL.

1. *Enormous action space*: $|\mathcal{V}| \sim 50000$ actions per step; standard exploration is infeasible.
2. *Long horizon*: $T \sim 100$–10000 tokens; credit assignment is hard.
3. *Pre-trained initialisation*: the LM is already a good policy; RL fine-tunes it.
4. *Mode collapse*: RL can collapse the LM to a narrow high-reward distribution (reward hacking).

LLMs as sequential decision processes. A large language model generating a sequence of tokens $a_1, a_2, \ldots, a_T$ can be formalised as an MDP:

- State s_t: the conversation history (prompt + tokens generated so far).
- Action a_t: the next token chosen from vocabulary $\mathcal{V}$.
- Transition: deterministic — $s_{t+1} = [s_t; a_t]$ (concatenate).
- Reward $r(s_T, a_T)$: a scalar quality signal assigned at the end of generation (terminal reward).

The terminal-reward structure means the MDP has extremely delayed, sparse rewards — the key technical challenge of RLHF (Reinforcement Learning from Human Feedback).

Reward model. A *reward model* $r_\phi(s)$ is trained from human preference comparisons: given two completions (y_1, y_2) to the same prompt x, a human annotator selects the preferred one. The Bradley-Terry model of pairwise preferences gives the loss:

$$\mathcal{L}_{\text{BT}}(\phi) = -\mathbb{E}_{(x,y_w,y_l)}\left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right]$$

where y_w is the preferred ("won") completion and y_l is the rejected ("lost") one.

RLHF pipeline. Three stages:

1. **SFT**: supervised fine-tune the LLM on demonstration data.
2. **RM**: train the reward model on human preference pairs.
3. **RL**: use PPO (or similar) to optimise the SFT policy against the RM, with a KL penalty $-\beta D_{\text{KL}}(\pi \,\|\, \pi_{\text{ref}})$ to prevent reward hacking.

106.2 Group Relative Policy Optimisation (GRPO)

Motivation. PPO requires a separate value function (critic) with as many parameters as the policy, doubling memory. GRPO (DeepSeek, 2024) eliminates the critic by estimating advantages from a group of sampled outputs.

GRPO objective. For prompt x, sample G outputs $\{y_i\}_{i=1}^G$ and score them with reward model r_ϕ:

$$\hat{A}_i = \frac{r_\phi(x, y_i) - \text{mean}_j(r_\phi(x, y_j))}{\text{std}_j(r_\phi(x, y_j))}$$

The advantage is normalised within the group (self-normalised baseline, no critic needed). Policy loss (with clipping as in PPO):

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{G}\sum_{i=1}^{G} \min\Big(\rho_i \hat{A}_i,\ \text{clip}(\rho_i, 1\pm\varepsilon)\hat{A}_i\Big) + \beta\, D_{\text{KL}}(\pi_\theta(y \mid x) \,\|\, \pi_{\text{ref}}(y \mid x))$$

where $\rho_i = \pi_\theta(y_i|x)/\pi_{\theta_{\text{old}}}(y_i|x)$.

Variance analysis. The group-normalised advantage has variance $\text{Var}[\hat{A}] = O(1/G)$ (decreasing with group size). For $G = 8$ (typical): variance reduction $8\times$ vs. single-sample baseline. The normalisation also automatically adapts to the reward scale, making hyperparameter tuning easier.

GRPO algorithm. Group Relative Policy Optimisation (Shao et al., 2024) is a variant of PPO designed for language model training that eliminates the value function critic (a separate neural network) to reduce memory and compute overhead.

Key idea. For each prompt q, sample G completions $\{o_1, \ldots, o_G\}$ from the current policy π_θ. Score each with a reward model: $r_i = r_\phi(q, o_i)$. Compute group-normalised advantages:

$$\hat{A}_i = \frac{r_i - \text{mean}(\{r_j\}_{j=1}^G)}{\text{std}(\{r_j\}_{j=1}^G)}$$

The advantage of a completion is its reward relative to the group mean, normalised by the group standard deviation. No bootstrapping, no value network required.

Policy update. Standard PPO clipped objective applied with the group advantages:

$$\mathcal{L}_{\text{GRPO}} = \mathbb{E}_i\left[\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{\text{old}}}(o_i|q)}\hat{A}_i,\ \text{clip}(\cdot, 1-\varepsilon, 1+\varepsilon)\hat{A}_i\right)\right] - \beta\, D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}})$$

DeepSeek-R1 and chain-of-thought. GRPO was used in training DeepSeek-R1 (2025) with a simple outcome reward: $r = 1$ if the final answer is correct, $r = 0$ otherwise. The model spontaneously developed chain-of-thought reasoning, backtracking, and self-correction behaviours — emergent from the RL process without any supervised chain-of-thought demonstration data.

GRPO vs PPO for LLMs

PPO requires a separate value network V_ϕ (same size as the policy model), doubling GPU memory. GRPO replaces V_ϕ with within-group normalisation, reducing memory by $\sim 50\%$ at the cost of higher variance in advantage estimates (batch size G instead of V_ϕ-based baseline).

106.3 Process Reward Models and Chain-of-Thought

Outcome vs. process rewards. An *outcome reward model* (ORM) scores only the final output $r(x, y)$. A *process reward model* (PRM) scores each reasoning step $r(x, y_{1:t})$, providing dense feedback.

PRM training (Lightman et al., 2023). Given a reasoning chain $y = (s_1, \ldots, s_K)$ (steps), collect human labels for correctness of each step: $\ell_k \in \{-, +\}$. Train $r_\phi(x, s_{1:k})$ to predict ℓ_k (binary cross-entropy). PRM reward: $r_t^{\text{PRM}} = r_\phi(x, y_{1:t}) - r_\phi(x, y_{1:t-1})$ (reward for step t). This provides credit assignment at the level of reasoning steps rather than whole answers.

MCTS for reasoning. Use the PRM as the value function in MCTS: select reasoning steps by $\text{UCT}(s, \text{step}) = Q(s, \text{step}) + c\,\hat{\pi}(\text{step}|s)\sqrt{N(s)}/(1+N(s, \text{step}))$, expand by sampling the next reasoning step from π_θ, and back up PRM scores. AlphaProof (DeepMind, 2024) applies this to mathematical theorem proving.

Example — GRPO on mathematical reasoning (DeepSeek-R1)

Task: solve competition-level math problems. Reward: $r = 1$ if final answer correct, $r = 0$ otherwise. Group size $G = 8$: sample 8 solutions per problem, normalise rewards within group. Outcome: model learns to generate chain-of-thought reasoning (intermediate steps) as a side effect of maximising correctness reward. The reasoning emerges from RL training, not from supervised imitation of reasoning traces. After RL fine-tuning: pass@1 on AIME 2024 improves from 5% (SFT baseline) to 52%.

106.4 Constitutional AI and RLAIF

RLAIF (RL from AI Feedback). Replace human preference labels with AI-generated labels. Claude or GPT-4 evaluates output pairs (y_1, y_2) according to a rubric and outputs a preference. Bradley-Terry model is then trained on these AI preferences. RLAIF scales annotation to millions of comparisons per day without human labellers.

Self-play fine-tuning (SPIN, Chen et al., 2024). Iterative self-improvement without any new human data: at iteration t, the policy π_t serves as both the "opponent" (generating negative examples) and the target being improved. Discriminator loss trains π_{t+1} to be distinguishable from π_t on human data. This is equivalent to maximum entropy IRL with the human dataset as the expert.

Constitutional AI mathematical guarantee. Under the assumption that the AI feedback model r_{AI} correctly ranks outputs according to a constitutional principle P with probability $1-\delta$, the RLHF policy π^* satisfies the principle with probability at least $1 - \delta \cdot T \cdot |\mathcal{V}|$ over a T-token generation. This bound is vacuous for long generations (requiring very high per-token accuracy), motivating PRM-based approaches.

Constitutional AI (CAI). Anthropic (Bai et al., 2022) introduced a self-supervised RL approach to alignment. Instead of human preference labels, CAI uses a *constitution* — a set of principles — to generate synthetic preference data.

CAI pipeline.

1. **SL-CAI**: given a harmful prompt, the model critiques its own response according to a constitutional principle, then revises it. Supervised fine-tuning on revised responses.
2. **RL-CAI**: the model scores pairs of responses using the constitution as the rubric (AI feedback), generating a preference dataset. A reward model is trained on this dataset. PPO with the AI-feedback RM replaces human-feedback RM.

RLAIF (RL from AI Feedback). The key insight: a sufficiently capable language model can provide reliable preference labels for most (prompt, response) pairs without human annotation. Lee et al. (2023) showed RLAIF matches human-feedback RL on summarisation benchmarks, reducing the bottleneck of expensive human annotation.

Direct Preference Optimisation (DPO). Rafailov et al. (2023) show that the RLHF objective (PPO with a KL-regularised reward model) has a closed-form solution that bypasses PPO

entirely:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta) = -\mathbb{E}_{(y_w,y_l)}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)\right]$$

DPO directly optimises the policy against preference data without a separate reward model or RL training loop, dramatically simplifying the alignment pipeline.

Part 107

Distributional Shift, Offline RL, and Conservative Methods

We develop the theory of offline RL in depth: the pessimism principle, in-distribution Q-learning, and the mathematical foundations of Conservative Q-Learning, TD3+BC, and Decision Transformer.

107.1 Offline RL: The Distributional Shift Problem

Offline dataset. Let $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^N$ be a fixed dataset collected by behaviour policy π_β (unknown). The offline RL goal: learn π maximising $J(\pi)$ without any additional environment interaction.

Why standard Q-learning fails offline. Consider the Bellman backup in Q-learning:

$$y_j = r_j + \gamma \max_{a'} Q_\theta(s_j', a')$$

The $\max_{a'}$ queries Q at actions (s_j', a^*) where $a^* = \arg\max_{a'} Q(s_j', a')$. If a^* was never in the dataset (out-of-distribution), $Q(s_j', a^*)$ is an unconstrained extrapolation of the neural network — typically overoptimistic. As this propagates through Bellman backups, Q-values for OOD actions diverge to $+\infty$, and the learned policy exploits them catastrophically.

Quantification. Let $\varepsilon_{\text{approx}}(s,a) = Q_\theta(s,a) - Q^{\pi_\beta}(s,a)$ be the approximation error. For $a \in \text{supp}(\pi_\beta(\cdot|s))$: $\varepsilon_{\text{approx}}$ is bounded by the generalisation error of the function approximator. For $a \notin \text{supp}(\pi_\beta(\cdot|s))$: $\varepsilon_{\text{approx}}$ is unbounded (no training data constraint). Standard Bellman backup $\max_{a'} Q(s', a')$ preferentially selects OOD actions.

The offline RL setting. In offline RL (also called batch RL), the agent has access only to a fixed dataset $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^N$ collected by some *behaviour policy* μ, without any additional interaction with the environment.

Distributional shift. The central challenge: the dataset was collected under μ, but we want to evaluate and improve a new policy $\pi \neq \mu$. When π visits state-action pairs (s,a) not covered by μ, the Q-function estimate $Q_\theta(s,a)$ is based on out-of-distribution (OOD) data — and Q-function errors at OOD points can compound catastrophically through the Bellman backup.

Distributional Shift Failure Mode

Suppose Q_θ overestimates $Q(s^*, a^*)$ at an OOD point due to function approximation error. The greedy policy $\pi(s) = \arg\max_a Q_\theta(s,a)$ will then select a^* at s^*. The Bellman backup propagates this overestimate backwards, creating a "runaway" overestimation cascade that is bounded by nothing in offline data.

Conservative Q-Learning (CQL). Kumar et al. (2020) add an explicit Q-value penalty to discourage large values at OOD actions:

$$\min_\theta \ \mathbb{E}_{(s,a)\sim\mathcal{D}}[(Q_\theta - y)^2] + \alpha\left(\mathbb{E}_{s\sim\mathcal{D},a\sim\pi}[Q_\theta(s,a)] - \mathbb{E}_{(s,a)\sim\mathcal{D}}[Q_\theta(s,a)]\right)$$

The second term pushes down Q-values under the learned policy (OOD actions) and pushes up values under the data distribution. This ensures $Q_\theta(s,a) \leq Q^\pi(s,a)$ in expectation — a conservative underestimate that prevents policy exploitation of OOD overestimations.

107.2 The Pessimism Principle

Theorem: Pessimism is near-optimal (Jin et al., 2021)

Let $\underline{Q}$ be a lower confidence bound on Q^*: $\underline{Q}(s,a) \leq Q^*(s,a)$ for all (s,a). The pessimistic policy $\hat{\pi} = \arg\max_\pi \mathbb{E}_\pi[\underline{Q}(s,\pi(s))]$ satisfies:

$$J(\pi^*) - J(\hat{\pi}) \leq \frac{2}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}}}[Q^*(s,\hat{\pi}(s)) - \underline{Q}(s,\hat{\pi}(s))]$$

The suboptimality gap is controlled by the tightness of the lower bound on the *pessimistic policy's* visitation.

Implication. If $\underline{Q}$ is tight for in-distribution state-action pairs (where $\hat{\pi}$ actually visits), the pessimistic policy is near-optimal. The lower bound can be loose on OOD pairs without affecting the bound — because the pessimistic policy avoids those regions by construction.

BEAR (Bootstrapping Error Accumulation Reduction) (Kumar et al., 2019). Constrains the policy to stay within the support of π_β via Maximum Mean Discrepancy (MMD):

$$\pi = \arg\max_\pi J(\pi) \quad \text{s.t.} \quad \text{MMD}(\pi(\cdot|s), \pi_\beta(\cdot|s)) \leq \varepsilon$$

where $\text{MMD}^2(\pi,\pi_\beta) = \mathbb{E}_{a,a'\sim\pi}[k(a,a')] - 2\mathbb{E}_{a\sim\pi,a'\sim\pi_\beta}[k(a,a')] + \mathbb{E}_{a,a'\sim\pi_\beta}[k(a,a')]$ with kernel k. This is tractably estimated from samples.

The pessimism principle. A unifying principle for offline RL: given uncertainty about $Q(s,a)$ at unobserved state-actions, act *pessimistically* — use a lower bound on Q. This prevents the policy from exploiting inaccurate Q-estimates.

Uncertainty quantification. Pessimism requires estimating the uncertainty of $Q_\theta(s,a)$. Methods include:

- **Ensemble disagreement**: train K Q-networks $\{Q_k\}$; uncertainty $= \text{std}(\{Q_k(s,a)\})$.
- **Count-based**: $u(s,a) = \beta/\sqrt{N_\mathcal{D}(s,a)}$ where $N_\mathcal{D}$ counts dataset visitations.
- **MOPO/MOReL**: train a dynamics model ensemble; pessimistic penalty = model ensemble disagreement propagated through simulated rollouts.

BEAR (Bootstrapping Error Accumulation Reduction). Kumar et al. (2019) constrain the learned policy to stay within the *support* of μ (the behaviour policy):

$$\max_\pi \; J_r(\pi) \quad \text{s.t.} \quad \mathbb{E}_{s\sim\mathcal{D}}[\text{MMD}(\pi(\cdot|s), \mu(\cdot|s))] \leq \epsilon$$

where MMD (Maximum Mean Discrepancy) is a kernel-based distributional distance. This ensures the policy does not select actions unsupported by data, avoiding OOD extrapolation.

Decision Transformer (DT). An alternative non-RL approach: train a sequence model (GPT-like transformer) to predict action given $(s_t, \text{return-to-go}_t, a_{t-1}, \ldots)$. At test time, condition on a high target return-to-go and the model "imitates" trajectories achieving that return. DT sidesteps Bellman error entirely but requires the offline dataset to contain high-return trajectories.

107.3 IQL, TD3+BC, and Offline Actor-Critic

IQL (Implicit Q-Learning, Kostrikov et al., 2021). Avoids ever evaluating Q at OOD actions by using *expectile regression* to compute an implicit maximum:

$$\mathcal{L}_V(\phi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[L_2^\tau(Q_{\bar{\psi}}(s,a) - V_\phi(s))\right]$$

where $L_2^\tau(u) = |\tau - \mathbf{1}(u<0)|u^2$ is the asymmetric quadratic loss and $\tau \in (0.5, 1)$. For $\tau \to 1$: $V_\phi(s) \to \max_{a:(s,a)\in\mathcal{D}} Q(s,a)$ (in-sample maximum). This gives the maximum over *in-sample* actions only, avoiding OOD queries. Policy extraction: advantage-weighted regression:

$$\pi_\theta = \arg\max_\theta \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\exp(\beta(Q_\psi(s,a) - V_\phi(s)))\log \pi_\theta(a|s)\right]$$

TD3+BC. The simplest offline RL algorithm achieving competitive performance:

$$\pi_\phi = \arg\max_\phi \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\lambda Q(s, \pi_\phi(s)) - (\pi_\phi(s) - a)^2\right]$$

with $\lambda = \alpha/\text{mean}_{(s,a)\in\mathcal{D}}|Q(s,a)|$. The first term maximises Q-value; the second term is behavioural cloning. The normalisation λ ensures the two terms are on the same scale.

Example — D4RL benchmark comparison

HalfCheetah-medium-v2 (locomotion with medium dataset):

Algorithm	Normalised Score
BC	42.6
CQL	44.0
IQL	47.4
TD3+BC	48.3
Decision Transformer	42.6
Expert policy	100.0

All methods significantly outperform BC (except DT, which uses return-conditioning rather than value optimisation).

Part 108

Continuous-Time RL and Neural ODEs

Many physical systems evolve in continuous time. We develop the Hamilton-Jacobi-Bellman equation as the continuous-time analogue of Bellman optimality, and connect RL to optimal control via the Pontryagin Maximum Principle.

108.1 Continuous-Time MDPs

Controlled diffusion. A continuous-time MDP is described by a stochastic differential equation:

$$ds_t = f(s_t, a_t)\,dt + \sigma(s_t, a_t)\,dW_t$$

where $f : \mathcal{S}\times\mathcal{A} \to \mathbb{R}^n$ is the drift, $\sigma : \mathcal{S}\times\mathcal{A} \to \mathbb{R}^{n\times m}$ is the diffusion coefficient, and W_t is an m-dimensional Wiener process. The reward accumulates continuously: $J(\pi) = \mathbb{E}\left[\int_0^\infty e^{-\rho t} r(s_t, a_t)\,dt\right]$ where $\rho > 0$ is the discount rate.

Discounted continuous-time return. The continuous-time discount factor $e^{-\rho t}$ corresponds to the discrete-time factor $\gamma = e^{-\rho\Delta t}$ in the limit $\Delta t \to 0$. For $\gamma = 0.99$ and $\Delta t = 0.01$: $\rho = -\log(0.99)/0.01 \approx 1.005$.

Continuous-time MDPs. In many physical systems (robotics, finance, neuroscience), state evolves continuously in time according to a stochastic differential equation (SDE):

$$ds_t = f(s_t, a_t)\,dt + \sigma(s_t, a_t)\,dW_t$$

where $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the drift, σ is the diffusion coefficient, and W_t is a standard Brownian motion. The corresponding MDP has state space $\mathcal{S} \subseteq \mathbb{R}^n$, action space $\mathcal{A} \subseteq \mathbb{R}^m$, and a continuous reward rate $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

Discounted return in continuous time.

$$J(\pi) = \mathbb{E}\left[\int_0^\infty e^{-\rho t}\, r(s_t, a_t)\, dt\right]$$

where $\rho > 0$ is the continuous discount rate. The relation to discrete discounting is $\gamma = e^{-\rho \Delta t}$, so $\rho = -\ln \gamma / \Delta t$.

Infinitesimal generator. For a function $v : \mathcal{S} \to \mathbb{R}$, the infinitesimal generator $\mathcal{A}^a$ of the controlled diffusion is:

$$\mathcal{A}^a v(s) = f(s,a)^\top \nabla_s v(s) + \frac{1}{2}\mathrm{tr}[\sigma\sigma^\top(s,a)\nabla_s^2 v(s)]$$

The first term is the directional derivative along the drift; the second is the Itô correction from the diffusion (Laplacian-like term).

Kolmogorov Backward Equation

The value function $V^\pi(s) = \mathbb{E}[\int_0^\infty e^{-\rho t} r(s_t, \pi(s_t))\, dt \mid s_0 = s]$ satisfies the PDE:

$$\rho V^\pi(s) = r(s, \pi(s)) + \mathcal{A}^{\pi(s)} V^\pi(s)$$

This is the continuous-time analogue of the Bellman equation.

Relationship to discrete time. As $\Delta t \to 0$, the discrete Bellman equation $V(s) = r(s,a)\Delta t + e^{-\rho \Delta t}\mathbb{E}[V(s')]$ converges to the backward equation above. This formal limit motivates many continuous-time RL algorithms (e.g., neural ODEs combined with RL, continuous-time actor-critic).

108.2 Hamilton-Jacobi-Bellman Equation

Hamilton-Jacobi-Bellman (HJB) Equation

The optimal value function $V^*(s) = \sup_\pi J(\pi; s)$ satisfies:

$$\rho V^*(s) = \sup_{a \in \mathcal{A}} \left\{ r(s,a) + \nabla_s V^*(s)^\top f(s,a) + \frac{1}{2}\mathrm{tr}\left[\sigma\sigma^\top(s,a)\,\nabla_s^2 V^*(s)\right] \right\}$$

The HJB is a nonlinear PDE. The three terms on the right:

- $r(s,a)$: instantaneous reward;
- $\nabla_s V^\top f(s,a)$: directional derivative of V along the drift (deterministic part);
- $\frac{1}{2}\mathrm{tr}[\sigma\sigma^\top \nabla^2 V]$: Ito correction from noise (second-order term).

In the deterministic case ($\sigma = 0$), HJB reduces to $\rho V^*(s) = \sup_a \{r(s,a) + \nabla V^* \cdot f(s,a)\}$.

Connection to discrete Bellman. Taking the limit $\Delta t \to 0$ in the discrete Bellman equation: $V^*(s) = \sup_a [r(s,a)\Delta t + e^{-\rho \Delta t}\mathbb{E}[V^*(s + f\Delta t + \sigma \Delta W)]]$, expanding to second order in Δt, and dividing by Δt gives exactly the HJB.

HJB equation. The Hamilton-Jacobi-Bellman (HJB) equation is the continuous-time analogue of the Bellman optimality equation. For a deterministic controlled system $\dot{s} = f(s,a)$:

$$\rho V^*(s) = \max_{a \in \mathcal{A}} \big[r(s,a) + \nabla_s V^*(s)^\top f(s,a) \big]$$

For the stochastic case $ds = f(s,a)\,dt + \sigma\,dW$:

$$\rho V^*(s) = \max_{a} \big[r(s,a) + \nabla_s V^*(s)^\top f(s,a) + \tfrac{1}{2}\mathrm{tr}(\sigma\sigma^\top \nabla_s^2 V^*(s)) \big]$$

The Itô correction $\frac{1}{2}\mathrm{tr}(\sigma\sigma^\top H)$ is the second-order term; for $\sigma \to 0$ the stochastic HJB reduces to the deterministic version.

Verification theorem. If $V^* \in C^2(\mathcal{S})$ solves the HJB and the maximising action:

$$\pi^*(s) = \arg\max_{a} \big[r(s,a) + \nabla_s V^*(s)^\top f(s,a) \big]$$

is well-defined, then V^* is the optimal value function and π^* is the optimal policy.

Example — Linear-Quadratic Regulator (LQR)

System: $\dot{s} = As + Ba$; cost: $r(s,a) = -(s^\top Q s + a^\top R a)$; discount $\rho \to 0$ (finite horizon). The HJB ansatz $V(s) = s^\top P s$ (quadratic) converts the PDE into the Riccati equation:

$$PA + A^\top P - PBR^{-1}B^\top P + Q = 0$$

whose solution $P \succ 0$ gives the optimal linear feedback $a^* = -R^{-1}B^\top P s$. This is the canonical closed-form RL solution for linear-quadratic systems.

Numerical solution. For general nonlinear systems, the HJB is solved:

- *Finite differences*: discretise $\mathcal{S}$ on a grid; exponential complexity in $\dim(\mathcal{S})$.
- *Value function approximation*: parametrise $V_\theta(s) \approx V^*(s)$ and minimise the HJB residual $\|HJB[V_\theta]\|^2$ ("physics-informed" neural networks for control).
- *Path integral control*: for systems where σ is large, the HJB admits a log-linear transformation connecting to a linear PDE solvable by Monte Carlo sampling.

108.3 Pontryagin Maximum Principle

Deterministic optimal control. For $ds_t = f(s_t, u_t)\,dt$ with $\sigma = 0$:

Pontryagin Maximum Principle (PMP)

Define the Hamiltonian $\mathcal{H}(s,u,p) = r(s,u) + p^\top f(s,u)$ where $p_t \in \mathbb{R}^n$ is the *co-state*. An optimal control $u^*(t)$ and trajectory (s_t^*, p_t^*) satisfy:

1. $\dot{s}^* = \nabla_p \mathcal{H}(s^*, u^*, p^*) = f(s^*, u^*)$ (state dynamics);
2. $\dot{p}^* = -\nabla_s \mathcal{H}(s^*, u^*, p^*) = -\nabla_s r(s^*, u^*) - (\nabla_s f(s^*, u^*))^\top p^*$ (co-state dynamics);

3. $u^*(t) = \arg\max_u \mathcal{H}(s^*(t), u, p^*(t))$ (optimality condition).

The co-state $p_t = \nabla_s V^*(s_t)$ is the gradient of the value function — the marginal value of an additional unit of state. The PMP converts the infinite-dimensional policy optimisation into a two-point boundary value problem (state BVP) solvable by shooting methods.

Relationship to RL. The actor-critic framework can be seen as a stochastic version of PMP: the critic $V_\phi(s)$ approximates $V^*(s)$; the gradient $\nabla_s V_\phi(s)$ approximates the co-state; the policy gradient $\nabla_\theta J$ mirrors the PMP optimality condition. SAC with reparametrised policy gradients computes $\nabla_a Q(s,a)|_{a=\mu_\theta(s)} \cdot \nabla_\theta \mu_\theta(s)$, which is exactly the chain-rule form of the PMP condition.

Example — LQR as special case

Linear dynamics: $\dot{s} = As + Bu$. Quadratic reward: $r(s,u) = -(s^\top Q s + u^\top R u)$, $Q, R \succ 0$. HJB with ansatz $V^*(s) = -s^\top P s$: leads to the Riccati equation $PA + A^\top P - PBR^{-1}B^\top P + Q = 0$. Optimal control: $u^*(s) = -R^{-1}B^\top P s$ (linear feedback). PMP: co-state $p = -2Ps$; optimality $\partial\mathcal{H}/\partial u = -2Ru + 2B^\top p = 0$ gives $u = R^{-1}B^\top p = -R^{-1}B^\top P s$. Consistent.

Part 109

Sample Complexity of Exploration: Lower Bounds and Algorithms

We derive fundamental sample complexity lower bounds for RL and present optimal algorithms. The Burn-in problem, the tabula rasa complexity, and the gap between sample efficiency of model-based vs. model-free methods.

109.1 Information-Theoretic Lower Bounds

The fundamental dilemma. Any learning algorithm must balance exploration (gathering information to identify the optimal policy) and exploitation (acting on current knowledge). The information needed is fundamentally limited by the structure of the MDP.

Minimax lower bound for episodic RL

For any algorithm $\mathcal{A}$ and constants $c_0 > 0$, there exists an MDP with S states, A actions, and episode length H such that with probability $\geq 1/2$:

$$\text{Regret}(K) \geq c_0 \sqrt{H \cdot S \cdot A \cdot K}$$

Proof sketch. Construct a "hard" MDP: a binary tree of depth H with A actions per node, where at the root a single "good" action has reward 1 and all others have reward 0. All transitions are uniform and the good action is at a random leaf. Any algorithm must identify the good action path, requiring $\Omega(\sqrt{SAK})$ episodes by reduction to binary hypothesis testing and Fano's inequality.

Separation: model-based vs. model-free. Tabular model-based RL (learn $\hat{\mathcal{P}}$, solve, execute) achieves the optimal $\tilde{O}(\sqrt{HSAK})$ regret. Model-free Q-learning has been shown to achieve the same asymptotic rate (Li et al., 2021), but with worse constants in practice.

Minimax lower bounds. Information-theoretic lower bounds quantify the minimum number of samples any algorithm requires to learn an ε-optimal policy.

Le Cam's method. To lower bound sample complexity, construct two MDPs $\mathcal{M}_1$ and $\mathcal{M}_2$ that are hard to distinguish but have different optimal policies:

$$\text{Sample complexity} \geq \frac{c}{\varepsilon^2} \cdot \frac{1}{D_{\text{TV}}(\mathcal{M}_1, \mathcal{M}_2)^2}$$

where D_{TV} is the total variation distance between trajectory distributions.

Lower bound for tabular RL (Azar et al., 2013). For any algorithm and any $\varepsilon > 0$:

$$\text{Sample complexity} \geq \Omega\left(\frac{|\mathcal{S}||\mathcal{A}|}{(1-\gamma)^3\varepsilon^2}\right)$$

The $(1-\gamma)^{-3}$ factor comes from: $(1-\gamma)^{-1}$ for the effective horizon, $(1-\gamma)^{-1}$ for the return scale, and $(1-\gamma)^{-1}$ for propagation depth.

Linear function approximation lower bounds. For feature dimension d:

$$\text{Sample complexity} \geq \Omega\left(\frac{d}{(1-\gamma)^3\varepsilon^2}\right)$$

This matches (up to log factors) the upper bound of LSVI-UCB (Jin et al., 2020), proving that optimism-based linear RL algorithms are near-minimax-optimal.

Hardness of nonlinear function approximation. Du et al. (2020) constructed MDPs where even with a perfect neural network representation, any algorithm requires exponentially many samples. This negative result shows that general nonlinear function approximation requires structural assumptions (e.g., low-rank transitions, block MDPs) for efficient RL.

109.2 The Burn-in Period

Sample complexity in the discounted setting. For an MDP with S states, A actions, discount γ, and target accuracy ε:

$$N_{\text{sample}} = \tilde{O}\left(\frac{SA}{(1-\gamma)^3\varepsilon^2}\right)$$

This is the number of samples needed to find an ε-optimal policy. The $(1-\gamma)^{-3}$ factor reflects three obstacles:

1. Mixing time $\sim 1/(1-\gamma)$: time for value information to propagate;
2. Effective horizon $\sim 1/(1-\gamma)$: episode length needed;
3. Per-sample variance $\sim 1/(1-\gamma)$: variance of return estimates.

Comparison to supervised learning. In SL with N i.i.d. samples, accuracy scales as $O(1/\sqrt{N})$. In RL: the effective number of independent samples is $N_{\text{eff}} \sim N \cdot (1-\gamma)^2$ (correlation within trajectories reduces effective sample count). This $(1-\gamma)^2$ penalty is fundamental and explains why RL requires many more samples than SL for comparable accuracy.

The burn-in problem. Model-free RL algorithms typically require a large number of environment steps before the learned policy becomes competitive with the optimal policy. This "burn-in" period has both information-theoretic and algorithmic sources.

Information-theoretic lower bounds on burn-in. Even an optimal algorithm must explore to gain information about $\mathcal{P}$ and $\mathcal{R}$. For a tabular MDP with $|\mathcal{S}|$ states, $|\mathcal{A}|$ actions, and horizon H:

$$\text{Sample complexity} \geq \Omega\left(\frac{|\mathcal{S}||\mathcal{A}|H^2}{\varepsilon^2(1-\gamma)^2}\right)$$

to learn an ε-optimal policy with constant probability. This matches (up to log factors) the upper bound achieved by algorithms like UcbVI (Azar et al., 2017).

Warm-start strategies. Several techniques reduce the effective burn-in:

- **Imitation learning**: initialise π_θ with behavioural cloning before RL training.
- **Hindsight Experience Replay (HER)**: relabel failed trajectories with the state actually reached as the goal, turning every failure into a learning signal for some goal.
- **Curriculum**: present easy tasks first (short horizons, dense rewards), then progressively increase difficulty.
- **Offline pre-training**: pre-train from an offline dataset before fine-tuning online.

Example — HER on Fetch robotic manipulation

Andrychowicz et al. (2017) showed that without HER, DQN/DDPG fails entirely on sparse-reward robotic pushing (the robot never reaches the goal by chance). With HER, the agent solves the task within $\sim 5 \times 10^6$ steps by treating each failed trajectory as a successful trajectory toward the state actually achieved.

109.3 Sample Efficiency in Deep RL

Data-augmented RL (DrQ). For pixel-based observations, apply random image augmentations (crop, translate, colour jitter) to training observations: $o_{\text{aug}} = \text{aug}(o)$. Train both actor and critic on augmented observations; soft Q-function target uses augmented next state. This effectively multiplies the dataset by the number of augmentation types — typically 10–100$\times$ improvement in sample efficiency on DMControl benchmarks.

SPR (Self-Predictive Representations) (Schwarzer et al., 2021). Auxiliary task: predict future latent states given current state + action sequence:

$$\mathcal{L}_{\text{SPR}} = -\sum_{k=1}^{K} \cos\big(g_\phi(f_\theta(z_t, a_t, \dots, a_{t+k-1})),\ \text{sg}(h_\xi(z_{t+k}))\big)$$

where sg is stop-gradient and h_ξ is a momentum encoder. This latent self-prediction forces the representation to encode the dynamics structure, providing sample efficiency gains of $\sim$ 2–3$\times$ over DQN on Atari.

Example — Sample efficiency comparison: Atari 100K

All methods trained on exactly 100K environment steps ($\approx$ 2 hours of game time):

Algorithm	Human-normalised score (%)
DQN (random agent)	0
DQN	2.3
DrQ	28.0
SPR	70.4
EfficientZero	194.3
Human	100.0

EfficientZero (model-based + MuZero + data augmentation) achieves superhuman performance in 100K steps — 400$\times$ fewer than DQN to human-level performance.

The sample efficiency gap. Deep RL algorithms require orders of magnitude more samples than humans to learn complex tasks. DQN requires ~ 50 million frames (~ 200 hours of gameplay) to reach human-level Atari performance; humans achieve similar performance in ~ 2 hours.

Sources of inefficiency.

1. **Credit assignment**: in tasks with long horizons and sparse rewards, attributing the reward to the correct action is difficult. The effective horizon is $1/(1-\gamma)$ steps.
2. **Exploration**: without structure, random exploration requires exponentially many steps to find rewards in large state spaces.
3. **Model-free learning**: the agent does not generalise across similar state-action pairs unless function approximation shares structure.
4. **Bootstrap variance**: each Bellman update is a noisy estimate; convergence requires many samples to average out noise.

Improving sample efficiency. Several algorithmic innovations address specific sources:

- **Data augmentation** (DrQ, RAD): apply random cropping, cutout, or other augmentations to observations; dramatically improves pixel-based RL.
- **MBPO**: short model rollouts from a learned model multiply effective dataset size $k\times$.
- **SPR** (Self-Predictive Representations): auxiliary loss predicts future latent representations, improving representation quality at no environment cost.
- **EfficientZero**: combines MCTS planning, self-supervised learning, and a world model; achieves human-level Atari performance in 2 hours of gameplay ($7\times$ less than DQN).

Data-efficiency frontier (2024). The field has made dramatic progress: algorithms like TD-MPC2 and EfficientZero achieve > 100 median human-normalised score on Atari-100k (only 100k environment frames) versus DQN's ~ 0.2 at the same budget.

PART 110

Representation Learning for RL

The quality of state representations directly determines sample efficiency and generalisation in RL. We cover bisimulation metrics, successor representations, contrastive methods, and the mathematical theory of representation learning objectives.

110.1 Bisimulation Metrics

Behavioural equivalence. Two states s, s' are *bisimilar* under policy π if they have identical immediate reward and identical distributions over equivalent next states: $s \sim s'$ iff $\mathcal{R}(s,a) = \mathcal{R}(s',a)$ for all a and $\mathcal{P}(\cdot|s,a) = \mathcal{P}(\cdot|s',a)$ for all a (where $\mathcal{P}$ maps to equivalence classes under $\sim$). This is circular (fixed-point definition).

Bisimulation metric (Ferns et al., 2004)

The bisimulation metric $d : \mathcal{S} \times \mathcal{S} \to [0, \infty)$ satisfies:

$$d(s,s') = \max_a \left[(1-c)|R(s,a) - R(s',a)| + c \cdot W_1(\mathcal{P}(\cdot|s,a), \mathcal{P}(\cdot|s',a); d)\right]$$

where W_1 is the Wasserstein-1 (earth mover's) distance under the metric d and $c \in (0, 1)$.

The bisimulation metric is the unique fixed point of this operator (contraction for $c < 1$). Crucially: $|V^\pi(s) - V^\pi(s')| \leq d(s, s') \cdot r_{\max}/(1-\gamma)$. States with small bisimulation distance have similar values — the metric captures value-relevant similarity.

Deep bisimulation (DBC) (Zhang et al., 2021). Learn a latent encoder $\phi_\theta(s)$ such that $||\phi_\theta(s) - \phi_\theta(s')||_2 \approx d^\pi_{\text{bisim}}(s, s')$:

$$\mathcal{L}_{\text{bisim}} = (||\phi_\theta(s) - \phi_\theta(s')||_2 - |R(s,a) - R(s',a)| - \gamma W_2(\hat{\mathcal{P}}(\cdot|s,a;\phi), \hat{\mathcal{P}}(\cdot|s',a;\phi)))^2$$

This loss trains the encoder to satisfy the bisimulation metric, removing value-irrelevant features (e.g., background clutter in image-based RL).

Bisimulation. Two states s and s' are *bisimilar* if they are behaviourally equivalent: same immediate reward and same distribution over next bisimilarity classes.

Bisimulation metric (Ferns et al., 2004)

The bisimulation metric $d : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the unique fixed point of:

$$F(d)(s, s') = \max_a \left[|r(s,a) - r(s',a)| + \gamma W_1(P(\cdot|s,a), P(\cdot|s',a); d)\right]$$

where $W_1(\mu, \nu; d) = \inf_{\kappa \in \Gamma(\mu,\nu)} \int d(x,y)\, d\kappa(x,y)$ is the Kantorovich (Wasserstein-1) distance between next-state distributions.

Bisimulation implies value similarity.

$$|V^\pi(s) - V^\pi(s')| \leq \frac{2r_{\max}}{(1-\gamma)^2}\, d_\pi(s, s')$$

where d_π is the policy-specific bisimulation metric. States close under d_π have similar values — a formal grounding for why bisimilarity is a useful state abstraction.

Deep bisimulation for control (DBC). Zhang et al. (2020) learn an encoder $z = \phi(s)$ such that $\|z - z'\|_1 \approx d(s, s')$ in bisimulation metric. The encoder is trained with:

$$\mathcal{L}_{\text{bisim}}(\phi) = \mathbb{E}_{s,s'}\left[\left(\|\phi(s) - \phi(s')\|_1 - |r(s) - r(s')| - \gamma W_1(\phi_\# P(\cdot|s), \phi_\# P(\cdot|s'))\right)^2\right]$$

This gives structured, reward-predictive representations without contrastive negative pairs.

110.2 Successor Representations

Successor representation (SR) (Dayan, 1993). Define the SR of state s under policy π:

$$M^\pi(s, s'') = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}[s_t = s''] \,\middle|\, s_0 = s\right]$$

$M^\pi(s, s'')$ is the discounted expected number of visits to s'' starting from s under π.

Value decomposition. Let $w(s) = \mathbb{E}_\pi[\mathcal{R}(s, \pi(s))]$ be the expected reward at state s. Then:

$$V^\pi(s) = \sum_{s''} M^\pi(s, s'')\, w(s'')$$

The SR $M^\pi(s,\cdot)$ encodes the *transition structure* and w encodes *reward*; they are separated. This enables rapid adaptation to new rewards: keep M^π fixed, update only w.

Deep SR networks. With neural features $\phi(s) \in \mathbb{R}^d$, approximate $M^\pi(s,\cdot) \approx \psi^\pi(s)^\top$: $V^\pi(s) \approx \psi^\pi(s)^\top w$ where w is a linear reward weight vector. SR loss: $\psi^\pi(s) = \phi(s) + \gamma \mathbb{E}_{s'\sim\pi}[\psi^\pi(s')]$. Trained by TD on successor features: $\psi^\pi(s) \leftarrow \psi^\pi(s) + \alpha(\phi(s) + \gamma\psi^\pi(s') - \psi^\pi(s))$.

Successor representation (SR). Dayan (1993) proposed factoring the value function as:

$$V^\pi(s) = \sum_{s'} M^\pi(s,s')\, r(s')$$

where $M^\pi(s,s') = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}[s_t = s'] \mid s_0 = s\right]$ is the *successor representation* — the discounted expected future occupancy of s' starting from s.

Bellman equation for SR.

$$M^\pi(s,s') = \mathbf{1}[s=s'] + \gamma \sum_{\tilde{s}} \mathcal{P}^\pi(\tilde{s}|s)\, M^\pi(\tilde{s},s')$$

In matrix form: $\mathbf{M}^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1}$, i.e., the *discounted state visitation matrix*.

Transfer learning with SR.

- If only the reward function changes ($r \to r'$) but the dynamics stay the same, the SR M^π is unchanged.
- New values: $V^\pi_{r'}(s) = \sum_{s'} M^\pi(s,s') r'(s')$ computed in $O(|\mathcal{S}|^2)$ (matrix-vector product), no relearning needed.
- This is the key advantage of SR over standard Q-functions for reward generalisation.

Deep SR. For large state spaces, approximate $M^\pi(s,\cdot) \approx M_\phi(s)$ with a neural network. Train with temporal difference on the SR:

$$M_\phi(s_t) \leftarrow M_\phi(s_t) + \alpha\left[\mathbf{e}_{s_t} + \gamma M_\phi(s_{t+1}) - M_\phi(s_t)\right]$$

where $\mathbf{e}_{s_t}$ is a one-hot vector. In the deep case, this uses the feature representation $\phi(s_t)$ instead of one-hot, giving the *predictive representation* (Jaderberg et al., 2017; Universal Value Functions).

110.3 Contrastive Representation Learning

CURL (Contrastive Unsupervised RL) (Laskin et al., 2020). Apply augmentation to observation pairs: $o_q = \text{aug}(o)$, $o_k = \text{aug}'(o)$ (different augmentations of same frame). Train encoder f_θ using MoCo-style contrastive loss:

$$\mathcal{L}_{\text{CURL}} = -\log \frac{\exp(z_q^\top W z_k)}{\exp(z_q^\top W z_k) + \sum_{j=1}^{N} \exp(z_q^\top W z_j^-)}$$

where $z_q = f_\theta(o_q)$, $z_k = f_\xi(o_k)$ (momentum encoder), W is a bilinear product matrix, and $\{z_j^-\}$ are negative examples from the queue. The encoder learns to make augmented versions of the same frame similar and different frames dissimilar.

Why contrastive losses work for RL. The contrastive objective encourages the encoder to capture information invariant to augmentation (e.g., game state) while discarding augmentation-specific features (e.g., crop offsets). This matches the desiderata for RL representations: encode what matters for value (task-relevant state features) and discard distractors.

Part 111

Connections: RL and Optimal Transport, Control Theory, and Thermodynamics

We explore deep mathematical connections between RL and other fields: the Wasserstein distance in distributional RL, KL-regularised RL as free energy minimisation, and the duality between value functions and entropy in thermodynamic RL.

111.1 Wasserstein Distances and Distributional RL

Wasserstein-p distance. For probability distributions μ, ν on $\mathbb{R}$:

$$W_p(\mu,\nu) = \left(\inf_{\gamma\in\Gamma(\mu,\nu)} \int |x-y|^p \, d\gamma(x,y)\right)^{1/p}$$

where $\Gamma(\mu,\nu)$ is the set of couplings (joint distributions with marginals μ and ν). For $p=1$: $W_1(\mu,\nu) = \int |F_\mu(x) - F_\nu(x)|\,dx$ (CDF difference; Kantorovich-Rubinstein duality). For $p=2$: W_2 is related to the optimal transport cost; defines the Wasserstein geometry on probability space.

Distributional Bellman contraction in W_p. The distributional Bellman operator: $\mathcal{T}Z(s,a) \overset{d}{=} r(s,a) + \gamma Z(s', \pi(s'))$, $s' \sim \mathcal{P}(\cdot|s,a)$.

Theorem

$\mathcal{T}$ is a γ-contraction in the maximal Wasserstein-p metric: $\bar{W}_p(\mathcal{T}Z_1, \mathcal{T}Z_2) \leq \gamma \bar{W}_p(Z_1, Z_2)$ where $\bar{W}_p(Z_1,Z_2) = \sup_{s,a} W_p(Z_1(s,a), Z_2(s,a))$.

The proof uses the coupling interpretation: for any couplings of $Z_1(s',a'), Z_2(s',a')$, the scaling by γ contracts distances. Unique fixed point Z^* exists by Banach.

111.2 KL-Regularised RL as Free Energy Minimisation

Variational free energy. In statistical mechanics, the free energy of a system at inverse temperature β is: $F = \mathbb{E}_q[E] - \beta^{-1}H(q) = \mathbb{E}_q[E] + \beta^{-1}D_{\mathrm{KL}}(q\|p_0)$ where E is energy, $H(q)$ is entropy, and $p_0 \propto e^{-\beta E}$ is the Gibbs distribution. Minimising free energy: $\arg\min_q F = p_0$ (the Gibbs measure).

Correspondence. In maximum entropy RL with temperature α:

$$\begin{aligned}
\pi^* &= \arg\max_\pi \left[\mathbb{E}_\pi[r(s,a)] + \alpha H(\pi)\right] \\
&= \arg\min_\pi \left[-\mathbb{E}_\pi[r(s,a)] - \alpha H(\pi)\right] \\
&= \arg\min_\pi \mathbb{E}_\pi[-r(s,a)/\alpha] + D_{\mathrm{KL}}(\pi\|\text{Uniform}) \\
&= \arg\min_\pi D_{\mathrm{KL}}(\pi\|\pi_0 \cdot e^{r/\alpha})
\end{aligned}$$

This is free energy minimisation with energy $E(a|s) = -r(s,a)/\alpha$ and $\pi_0 = \text{Uniform}$. The temperature $\alpha = 1/\beta$ plays the role of thermodynamic temperature.

KL-control (Todorov, 2007). General KL-regularised RL: $J(\pi) = \mathbb{E}_\pi[r] - \alpha D_{\mathrm{KL}}(\pi\|\pi_0)$ where π_0 is a *prior policy* (not necessarily uniform). Optimal policy: $\pi^* \propto \pi_0 \cdot e^{r/\alpha}$. This generalises

maximum entropy RL to arbitrary reference policies π_0, which appears directly in RLHF ($\pi_0 = \pi_{\text{ref}}$).

KL-regularised RL. The MaxEnt RL objective can be written as:

$$\pi^* = \arg\min_{\pi} \mathbb{E}_{s\sim d^\pi}\left[D_{\text{KL}}\big(\pi(\cdot|s) \,\|\, e^{Q^*(s,\cdot)/\alpha}/Z(s)\big)\right]$$

This frames RL as *approximate inference*: the optimal policy is the posterior over actions given the "prior" $e^{Q^*/\alpha}$ (the Boltzmann distribution over Q-values). The temperature α controls the softness of the posterior.

Free energy interpretation. In statistical physics, the free energy $F = U - TS$ (internal energy minus temperature times entropy) is minimised at equilibrium. In KL-regularised RL:

$$\mathcal{F}(\pi) = -J_r(\pi) - \alpha\mathcal{H}(\pi) = \mathbb{E}_\pi[-r(s,a) + \alpha \log \pi(a|s)]$$

Minimising $\mathcal{F}$ simultaneously maximises reward and entropy, reproducing the MaxEnt RL objective.

Duality with inference. The connection to variational inference is exact:

- **Evidence**: $\log Z = \log \mathbb{E}_{a\sim\mu}[e^{Q(s,a)/\alpha}]$ (log-partition function).
- **ELBO**: $\mathbb{E}_\pi[Q(s,a)/\alpha] - D_{\text{KL}}(\pi \,\|\, \mu) \le \log Z$.
- Maximising the ELBO over π gives $\pi^*(a|s) \propto \mu(a|s)e^{Q(s,a)/\alpha}$, the MPO update.

Maximum a Posteriori Policy Optimisation (MPO, Abdolmaleki et al., 2018) exploits this duality to derive E-step (policy improvement) and M-step (policy fitting) updates, providing a principled EM algorithm for RL.

Example — Soft Q-learning as variational inference

Haarnoja et al. (2017) show that soft Q-learning solves: $\min_\pi F_{KL}(\pi) = \min_\pi \mathbb{E}[D_{\text{KL}}\big(\pi(\cdot|s_t) \,\|\, e^{(r+\gamma V)/\alpha}/Z\big)]$ at every state s_t. The SAC actor update is the practical stochastic gradient implementation of this, using the reparameterisation trick to backpropagate through the stochastic policy.

111.3 Value Functions as Log-Partition Functions

Soft value function as free energy.

$$V^{\text{soft}}(s) = \alpha \log \int_{\mathcal{A}} \pi_0(a|s)\, e^{Q^{\text{soft}}(s,a)/\alpha}\, da = \alpha \log Z(s)$$

This is the log-partition function $\log Z(s)$ of the Gibbs distribution over actions. The partition function $Z(s) = \mathbb{E}_{\pi_0}[e^{Q/\alpha}]$ is the (unnormalised) moment generating function of Q.

Value estimation via importance sampling. $V^{\text{soft}}(s) = \alpha \log \mathbb{E}_{a\sim\pi_0}\left[e^{Q(s,a)/\alpha}\right]$. For high temperature ($\alpha \to \infty$): $\log \mathbb{E}[e^{X/\alpha}] \approx \mathbb{E}[X]/\alpha + O(1/\alpha^2)$, giving $V \to \mathbb{E}_{\pi_0}[Q]$. For low temperature ($\alpha \to 0$): $\alpha \log \mathbb{E}[e^{X/\alpha}] \to \max_a Q(s,a)$ (Laplace principle).

Diffusion models as RL. Denoising diffusion models can be viewed as maximum entropy RL: the forward process is a fixed prior policy (Gaussian noise addition); the reverse process is the learned policy. DDPO (Black et al., 2023) explicitly frames image generation as RL

and fine-tunes diffusion models with PPO to maximise a reward function (e.g., image-text alignment, aesthetic quality).

Value functions as log-partition functions. The soft value function in MaxEnt RL:

$$V^*_{\text{soft}}(s) = \alpha \log \sum_a \exp\left(\frac{Q^*_{\text{soft}}(s,a)}{\alpha}\right) = \alpha \log Z_\alpha(s)$$

is the log-partition function of the Boltzmann distribution over actions. This connection to statistical physics is not coincidental: MaxEnt RL is formally equivalent to the computation of free energy in a statistical system where the Hamiltonian is $-Q(s,a)/\alpha$.

Consequences.

- **Compositionality**: for tasks with reward $r_{\text{new}} = r_1 + r_2$ (sum of two learned reward components), the log-partition function satisfies: $Z_{\text{new}} \propto Z_1 \cdot Z_2$ (product of experts). Soft values can be combined without retraining (Haarnoja et al., 2018; composable policies).
- **Connection to energy-based models**: $Q_{\text{soft}}(s,a) = -\mathcal{E}(s,a)$ where $\mathcal{E}$ is an energy function. Policy optimisation becomes energy minimisation.
- **Variational gap**: by Jensen's inequality, $V_{\text{soft}} \geq \mathbb{E}_\pi[Q] + \alpha\mathcal{H}(\pi)$ with equality iff π is the optimal softmax policy — a tight variational bound.

Control as inference (Levine, 2018). The entire RL problem can be reframed as inference in a graphical model with binary "optimality" variable $\mathcal{O}_t$: $P(\mathcal{O}_t = 1 | s_t, a_t) = \exp(r(s_t,a_t)/\alpha)$. The posterior $P(\tau|\mathcal{O}_{0:T} = 1)$ over optimal trajectories is the MaxEnt policy distribution. Variational inference in this model recovers soft Q-learning and SAC.

PART 112

Open Problems and Research Frontiers

We survey the major open problems in RL theory and practice: sample efficiency, reward misspecification, generalisation, causal RL, and the foundations of artificial general intelligence through RL.

112.1 Sample Efficiency Gap

Human vs. machine sample efficiency. Humans learn complex tasks (driving, chess, language) in $\sim 10^3$–10^6 samples; deep RL agents require $\sim 10^7$–10^9 samples for comparable performance. The gap comes from multiple sources:

1. *Prior knowledge*: humans have powerful inductive biases (physics, causality, object permanence) that RL agents must learn from scratch.
2. *Structured exploration*: humans use systematic strategies (divide and conquer, analogical reasoning) rather than random exploration.
3. *Transfer*: humans leverage past experience across tasks; most RL agents start from scratch.
4. *Causal understanding*: humans build causal models that generalise; RL agents learn correlations.

Theoretical gap. The best known lower bound for tabular RL: $\tilde{\Omega}(\sqrt{SAK})$ regret. Best upper

bound: $\tilde{O}(\sqrt{H^3SAK})$ (UCBVI). Gap of $H^{3/2}$ in the H factor. For function approximation: matching upper and lower bounds exist only for linear models; the nonlinear case is largely open. For deep RL: no non-vacuous sample complexity bounds.

The sample efficiency gap (empirical). Despite decades of progress, deep RL lags behind human and biological learning in sample efficiency on most benchmarks:

- **Atari**: human-level performance requires ∼200 hours (DQN) vs ∼15 minutes (human).
- **Dexterous manipulation**: state-of-the-art systems require 10^7–10^8 environment steps vs. human children learning new grasps in $\sim$ 10–100 attempts.
- **MuJoCo locomotion**: SAC achieves near-optimal gaits in $\sim 10^6$ steps; humans learn to walk in $\sim 10^4$ practise steps (in early childhood, over months).

Why humans are more efficient. Several hypotheses:

1. **Strong priors**: evolution and development give humans rich inductive biases about physics, object permanence, and goal structures.
2. **Structured world models**: humans learn compositional models that generalise across situations; deep RL agents typically learn flat lookup tables (with function approximation).
3. **Meta-learning**: humans have a lifetime of prior tasks; RL algorithms typically start from scratch.
4. **Social learning**: humans learn by observing others; most RL agents do not.

Implications for research. Closing the gap requires: (1) better inductive biases (compositionality, hierarchy, causality in the model architecture), (2) more structured exploration (directed, goal-conditioned), (3) world-model-based methods that leverage internal simulation, and (4) continual/lifelong learning across tasks.

112.2 Reward Misspecification and Goodhart's Law

Goodhart's Law. "When a measure becomes a target, it ceases to be a good measure." In RL: when r_ϕ (learned reward model) becomes the optimisation target, the optimal policy $\pi^*_{r_\phi}$ diverges from the optimal policy $\pi^*_{r^*}$ under the true reward r^*.

Formalisation. Let $r_\phi = r^* + \varepsilon$ (reward model error). The policy error:

$$J_{r^*}(\pi^*_{r^*}) - J_{r^*}(\pi^*_{r_\phi}) \leq \frac{2\,\|\varepsilon\|_{d^{\pi^*_{r_\phi}}}}{1-\gamma}$$

where $\|\varepsilon\|_d = \mathbb{E}_{(s,a)\sim d}|\varepsilon(s,a)|$ is the expected reward error under the exploiting policy. The key insight: the error is evaluated under $d^{\pi^*_{r_\phi}}$, which concentrates on OOD regions where ε may be arbitrarily large. Reward hacking is precisely this: finding OOD states where r_ϕ disagrees with r^*.

Approaches: (1) KL penalty: $r_{\text{total}} = r_\phi - \beta D_{\text{KL}}(\pi\|\pi_{\text{ref}})$ limits how far the policy can exploit r_ϕ. (2) Ensemble reward models: disagreement between ensemble members detects OOD exploitation. (3) Constitutional AI: explicit constraints reduce the set of exploitable dimensions.

Reward misspecification. In practice, the reward function is designed by engineers, not nature. Goodhart's Law: "When a measure becomes a target, it ceases to be a good measure." In RL terms: if r is a proxy for the true objective r^*, optimising r exactly may catastrophically fail to optimise r^*.

Examples of misspecification.

- **Reward hacking**: a boat-racing agent discovered it could accumulate more reward by spinning in circles hitting the same boost pads, never finishing the race (OpenAI, 2016).
- **Specification gaming**: a grasping robot learned to pick up blocks by exploiting camera positioning rather than actual grasping.
- **Distributional shift**: a reward learned from human feedback in a narrow distribution may extrapolate badly to out-of-distribution states (reward model overoptimisation in RLHF).

KL penalty in RLHF. To prevent reward hacking in language models, a KL penalty is added:

$$r_{\text{total}}(\pi) = r_\phi(s,a) - \beta\, D_{\text{KL}}(\pi(\cdot|s) \,\|\, \pi_{\text{ref}}(\cdot|s))$$

where π_{ref} is the supervised fine-tuned (SFT) reference model and β controls the deviation allowed. The optimal tradeoff: at $\beta = 0$ the model maximally exploits r_ϕ; at $\beta \to \infty$ the model never deviates from the SFT policy.

KL-reward Pareto frontier. Gao et al. (2022) empirically measure the gold-reward vs. KL trade-off in RLHF, finding that reward model score peaks at a finite KL budget and then decreases — direct evidence of overoptimisation. The optimal β lies at the Pareto frontier between fidelity to the base model and reward maximisation.

112.3 Causal Reinforcement Learning

Structural causal models. The environment's data-generating process is described by a DAG $\mathcal{G}$ with variables $\{X_i\}$ and structural equations $X_i := f_i(\text{Pa}_i, U_i)$ where Pa_i are parents and U_i are exogenous noise. An *intervention* $do(X_i = x)$ cuts all incoming edges to X_i and sets $X_i = x$.

Causal RL. Standard RL learns observational distributions $P(s_{t+1} \mid s_t, a_t)$. Causal RL learns interventional distributions $P(s_{t+1} \mid do(a_t))$, which is what the agent actually controls. In non-Markovian environments (with confounders), the two differ: actions causally set a_t, while observations come from a confounded mixture.

Counterfactual reasoning. "What return would I have gotten if I had taken action a' instead of a?" $Q^{CF}(s,a',\tau) = \mathbb{E}[G \mid s_t = s, do(a_t = a'), U = u_\tau]$ where u_τ are the exogenous variables realised in trajectory τ. Counterfactual advantage: $A^{CF}(s,a',a) = Q^{CF}(s,a') - Q^{CF}(s,a)$ uses the same noise realisation, reducing variance vs. standard advantage.

Causal RL. Standard RL conflates correlation and causation: an agent may learn to exploit spurious correlations that do not generalise across environments. Causal RL augments the MDP with a *structural causal model* (SCM) over state variables.

SCM formalism. An SCM $\mathcal{G} = (\mathbf{V}, \mathbf{E}, \mathcal{F})$ consists of:

- Observed variables $V_i \in \mathbf{V}$.
- Directed edges $(V_j \to V_i) \in \mathbf{E}$ (causal influences).

- Structural equations $V_i = f_i(\text{pa}(V_i), U_i)$ where U_i are independent noise terms.

The *do-calculus* (Pearl, 2009) distinguishes observational conditioning $P(Y|X = x)$ from interventional distribution $P(Y|\text{do}(X = x))$ (physically setting $X = x$ regardless of its causes).

Invariant Risk Minimisation (IRM). For RL across multiple environments $e \in \mathcal{E}$:

$$\min_{\theta} \sum_{e} \mathcal{L}^e(\theta) \quad \text{s.t.} \quad \theta \in \arg\min_{\bar{\theta}} \mathcal{L}^e(\bar{\theta}) \; \forall e$$

The constraint enforces that θ is simultaneously optimal across all environments, selecting only invariant (causal) features rather than environment-specific correlations.

Causal discovery in RL. Agents can actively intervene (take actions) to distinguish causal from spurious features — a key advantage of RL over passive observational causal inference. Recent work (Ke et al., 2021) learns a causal graph online via intervention-based exploration, then uses the graph to generalise policy across novel environments with different confounders.

112.4 Universal Reinforcement Learning and AI Safety

AIXI. Hutter (2005) proposed AIXI as the universal RL agent: it takes Bayesian expectation over all computable environments weighted by their Kolmogorov complexity K:

$$a_t^* = \arg\max_{a_t} \sum_{s_{t+1}} \sum_{\mu \in \mathcal{M}} 2^{-K(\mu)} P_\mu(s_{t+1} \mid s_{1:t}, a_{1:t}) V_\mu^*(s_{t+1})$$

AIXI is provably optimal against any computable environment but is incomputable (requires enumeration of all Turing machines). Approximations: MC-AIXI-CTW uses context tree weighting as the model class; AIXI-tl truncates to time-bounded programs.

Reward and value alignment. The alignment problem in AI: how to specify a reward function r such that π_r^* aligns with human values r^* (which are complex, partially specified, and evolving). Key challenges:

- *Reward underspecification*: many reward functions r have $\pi_r^* \approx \pi_{r^*}^*$ on the training distribution but diverge on novel states (Goodhart).
- *Value extrapolation*: human values are inconsistent, context-dependent, and hard to elicit.
- *Corrigibility*: we want agents that allow value correction; but a sufficiently capable agent has incentives to resist correction (convergent instrumental goals).

Seldonian algorithms (Thomas et al., 2019) provide safety guarantees during deployment: π is returned only if there exists a test set evaluation satisfying $P(J^C(\pi) \leq 0) \geq 1 - \delta$. Uses concentration inequalities on held-out data to guarantee behavioural constraints with high probability — a step toward RL with formal safety proofs.

PART 113

Extended Examples, Algorithm Implementations, and Case Studies

A collection of detailed worked examples, complete algorithm implementations, and end-to-end case studies connecting theory to practice.

113.1 Complete Q-Learning Implementation

Full tabular Q-learning algorithm with convergence analysis. We trace through a complete example on the FrozenLake-v1 environment (4×4 grid).

Q-Learning pseudocode (complete)

Inputs: MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$; learning rate α; exploration ε; episodes N.
Initialise: $Q(s,a) \leftarrow 0$ for all (s,a); set $t \leftarrow 0$.

For episode $k = 1, \ldots, N$:

1. Reset environment; observe s_0.
2. **For** step $t = 0, 1, 2, \ldots$ until terminal:
   (a) Select a_t: with prob. ε sample uniform random, else $\arg\max_a Q(s_t, a)$.
   (b) Execute a_t; observe r_t, s_{t+1}, done flag.
   (c) Compute TD error: $\delta_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a') \cdot (1 - \text{done}) - Q(s_t, a_t)$.
   (d) Update: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta_t$.
3. Decay $\varepsilon \leftarrow \max(\varepsilon_{\min}, \varepsilon \cdot \varepsilon_{\text{decay}})$.

Return: Optimal policy $\pi^*(s) = \arg\max_a Q(s,a)$.

Example — FrozenLake-v1: training trace

4×4 grid, stochastic transitions (slippery ice). $\gamma = 0.99$, $\alpha = 0.1$, $\varepsilon_0 = 1.0$, $\varepsilon_{\min} = 0.01$, decay$= 0.995$.

Q-table at episode 1 (random): Start $s = 0$, action `RIGHT`=1, slip to $s = 4$, $r = 0$. $\delta = 0 + 0.99 \times 0 - 0 = 0$. No update yet.
Q-table after 100 episodes: Most Q values are 0 (goal rarely reached). $Q(14, \text{DOWN}) \approx 0.05$ (near goal).
Q-table after 5000 episodes: $Q(0, \text{DOWN}) \approx 0.42$; $Q(4, \text{RIGHT}) \approx 0.51$; $Q(8, \text{RIGHT}) \approx 0.63$; $Q(9, \text{DOWN}) \approx 0.71$; $Q(10, \text{DOWN}) \approx 0.80$; $Q(14, \text{RIGHT}) \approx 0.91$. Learned path: $0 \to 4 \to 8 \to 9 \to 10 \to 14 \to 15$ (goal). Success rate $\sim 74\%$ per episode.
Convergence analysis: After N episodes, each (s,a) pair visited $\sim N \cdot (1-\varepsilon_t) \cdot d^\pi(s) + \varepsilon_t / |\mathcal{A}|$ times. With $|\mathcal{S}| = 16$, $|\mathcal{A}| = 4$: ~ 64 parameters converge in ~ 5000 episodes.

Q-table visualisation. Define the value function as $V(s) = \max_a Q(s,a)$:

$$\begin{pmatrix} 0.42 & 0.00 & 0.32 & 0.00 \\ 0.51 & \text{H} & 0.22 & \text{H} \\ 0.63 & 0.71 & 0.80 & \text{H} \\ \text{H} & 0.90 & 0.91 & \text{G} \end{pmatrix}$$

(H = hole, G = goal). Values decrease as the agent gets further from the goal, forming a value landscape.

Q-Learning implementation. We implement tabular Q-learning for the CliffWalking environment (a 4×12 grid with a cliff along the bottom row, reward -100 for falling off, -1 per step).

Example — Complete Python pseudocode

```
import numpy as np

# Hyperparameters
alpha = 0.1;  gamma = 0.99;  epsilon = 0.1
n_episodes = 500

Q = np.zeros((48, 4))  # 48 states, 4 actions

for ep in range(n_episodes):
    s = env.reset()
    done = False
    while not done:
        # epsilon-greedy
        if np.random.rand() < epsilon:
            a = np.random.randint(4)
        else:
            a = np.argmax(Q[s])

        s_next, r, done, _ = env.step(a)

        # Q-learning update
        td_target = r + gamma * np.max(Q[s_next]) * (not done)
        Q[s, a] += alpha * (td_target - Q[s, a])
        s = s_next
```

Key design choices.

- ϵ-greedy with $\epsilon = 0.1$ maintains exploration throughout training. An annealing schedule ($\epsilon = 1/(1 + \text{ep}/100)$) often improves asymptotic performance.
- The $(1 - \text{done})$ mask correctly handles terminal states: $V(\text{terminal}) = 0$.
- For CliffWalking, Q-learning converges to the optimal (short but risky cliff-edge path) while SARSA converges to the safer inland path — a direct consequence of on-policy vs off-policy learning.

Convergence diagnostics. Plot the episode return and the Bellman residual $\delta_{\max} = \max_{s,a} |Q(s,a) - r(s,a) - \gamma \max_{a'} Q(s',a')|$ over training. Convergence is indicated by: (i) episode return stabilising near optimal; (ii) Bellman residual decaying toward zero. In stochastic environments, residual never reaches zero; target is $\delta_{\max} < \varepsilon/(1-\gamma)$ for desired ε-optimality.

113.2 Policy Gradient on CartPole

REINFORCE with baseline. CartPole-v1: state $s = (\theta, \dot{\theta}, x, \dot{x})$, actions $\{L, R\}$, episode ends when pole falls ($|\theta| > 0.21$ rad) or $t > 500$.

Policy network: 3-layer MLP: $4 \to 64 \to 64 \to 2$, softmax output. **Baseline network:** $4 \to 64 \to 64 \to 1$ (value function V_ϕ).

REINFORCE update equations:

$$\delta_t = G_t - V_\phi(s_t), \quad \theta \leftarrow \theta + \alpha \sum_t \gamma^t \delta_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

$$\phi \leftarrow \phi - \alpha_V \sum_t (G_t - V_\phi(s_t))^2$$

Example — CartPole learning curve

Training with $\alpha = 0.001$, $\alpha_V = 0.005$, $\gamma = 0.99$, batch $= 10$ episodes:
Episode 100: mean reward ≈ 28 (random baseline: 12).
Episode 500: mean reward ≈ 120 (pole balancing improving).
Episode 1000: mean reward ≈ 350 (near-optimal: most episodes last $300+$ steps).
Episode 2000: mean reward $\approx 485 \pm 25$ (consistently > 400 steps).

Without baseline: episode 2000 reward $\approx 380 \pm 80$ (higher variance, worse convergence). Variance ratio (with/without baseline) $\approx 0.3/1.0$ — consistent with theory ($\approx 1 - R^2(G, V)$).

Gradient variance. Empirically, after training for 1000 episodes: $\text{Var}[\sum_t G_t \nabla \log \pi] / \text{Var}[\sum_t (G_t - V_\phi(s_t)) \nabla \log \pi] \approx 4.2$. Baseline reduces variance by $4.2\times$, consistent with the explained variance $R^2 \approx 0.76$ of V_ϕ.

113.3 PPO on MuJoCo

Hopper-v4 continuous control: 11-dimensional state, 3-dimensional action in $[-1, 1]^3$, maximum 1000 steps.

PPO architecture:

- Actor: $11 \to 64 \to 64 \to 6$ (mean and log-std of Gaussian), tanh-squashed.
- Critic: $11 \to 64 \to 64 \to 1$.

Hyperparameters: $\gamma = 0.99$, $\lambda = 0.95$ (GAE), $\varepsilon = 0.2$ (clip), $c_1 = 0.5$, $c_2 = 0.01$, $K = 10$ epochs, minibatch $= 64$.

Example — PPO training curve: Hopper-v4

Collecting $N = 2048$ steps per update, then running $K = 10$ epochs:

Update 50 (≈ 100k steps): mean episode reward ≈ 1100.
Update 100 (≈ 200k steps): mean episode reward ≈ 2000.
Update 200 (≈ 400k steps): mean episode reward ≈ 2800 (near expert).
Expert reward ≈ 3500.

KL divergence between old and new policy stays in $[0.01, 0.04]$ — clipping is effective. Value function explained variance: increases from 0.3 at update 10 to 0.85 at update 200. Entropy: decreases from 2.1 (near-uniform) to 0.3 (concentrated on good actions).

Clip fraction. Fraction of timesteps where the ratio $r_t(\theta)$ is clipped:

$$\text{clip_frac} = \frac{1}{NT} \sum_t \mathbf{1}\left[r_t(\theta) \notin [1-\varepsilon, 1+\varepsilon]\right]$$

Good training: clip_frac ≈ 0.1–0.2. Too high (> 0.4): learning rate too large. Too low (< 0.02): underutilising data.

113.4 Deep Dive: Soft Actor-Critic Hyperparameters

Effect of temperature α on SAC.

Temperature α	Final reward	Training speed	Stability
0.01 (too low)	1800	Fast initially	Unstable, collapses
0.2 (good)	3200	Steady	Stable
1.0 (too high)	2100	Slow	Stable but suboptimal
Auto-tuned	3100	Steady	Stable

Auto-tuning (target entropy $= -|A| = -3$) achieves near-optimal performance robustly without hyperparameter search.

Replay buffer size. With replay buffer $|\mathcal{D}|$:

- $|\mathcal{D}| = 10^4$: only recent data, near on-policy. Low variance but high bias (distribution shift).
- $|\mathcal{D}| = 10^6$: diverse data from full training history. Low bias, higher variance.

For Hopper-v4: $|\mathcal{D}| = 10^6$ achieves 3200 reward vs. 2400 for $|\mathcal{D}| = 10^4$.

113.5 Multi-Agent Example: Cooperative Navigation

MPE Cooperative Navigation (Mordatch & Abbeel, 2018): $N = 3$ agents, $M = 3$ landmarks. Agents must cover all landmarks; reward = negative sum of distances to nearest landmark. Collision penalty: -1 per pair of agents within distance 0.1.

MAPPO implementation. Shared actor network (parameter sharing): each agent receives its own observation $o_i = (x_i, v_i, \{x_j - x_i\}_{j \neq i}, \{l_k - x_i\}_{k=1}^{M})$. Centralised critic receives global state $(x_1, \ldots, x_N, l_1, \ldots, l_M, v_1, \ldots, v_N)$.

Example — MAPPO convergence

After 10M steps:

Algorithm	Mean reward	Collision rate
Random	-8.3	0.42
Independent PPO	-3.1	0.18
MAPPO (centralised critic)	-1.8	0.04
QMIX	-1.6	0.03

The centralised critic allows MAPPO to coordinate agent behaviour (assign agents to different landmarks), achieving near-optimal coverage with few collisions.

Multi-agent cooperative navigation. We consider $N = 3$ agents in a $2D$ continuous space with 3 landmarks. Each agent must navigate to a distinct landmark; the global reward is $r = -\sum_i d(a_i, \ell_i)$ (total distance), shared equally among agents.

MADDPG implementation. Each agent i has:

- Policy $\pi_{\theta_i}(a_i | o_i)$: actor conditioned on local observation o_i.

- Centralised Q-function $Q_{\phi_i}(o_1, \ldots, o_N, a_1, \ldots, a_N)$: critic with global information.

The critic update (centralised):

$$y = r_i + \gamma Q_{\phi_i^-}(o', \pi_{\theta_1}(o_1'), \ldots, \pi_{\theta_N}(o_N'))$$

The actor update (decentralised):

$$\nabla_{\theta_i} J = \mathbb{E}\left[\nabla_{a_i} Q_{\phi_i}(o, a_1, \ldots, a_N) \cdot \nabla_{\theta_i} \pi_{\theta_i}(o_i)\right]$$

Emergent coordination. With MADDPG, agents learn to avoid collisions (negative reward for overlap) and coordinate landmark assignments without explicit communication. Coordination emerges from the centralised training: each agent's critic sees all other agents' actions and can learn the implicit assignment structure.

Communication protocols. DIAL (Foerster et al., 2016) and CommNet allow differentiable message passing between agents. Agent i broadcasts $m_i = g_{\psi_i}(h_i)$ (a learned function of its hidden state); all messages are aggregated (mean, attention) and fed into each agent's next decision step, enabling learned communication under partial observability.

113.6 Convergence Diagnostics in Practice

Key metrics for RL training. Beyond episode reward, these diagnostics indicate healthy training:

1. **Explained variance of value function**: $\text{EV} = 1 - \text{Var}[G - V]/\text{Var}[G]$. Should increase toward 1.
2. **KL divergence** between old and new policy: should stay in $[0.01, 0.1]$ for PPO.
3. **Entropy**: should decrease gradually (exploration declines as policy specialises).
4. **Value loss**: should decrease then plateau (critic converging).
5. **Gradient norm**: should be bounded (gradient clipping at 0.5–5.0 is common).
6. **Actor loss**: should decrease overall but may be noisy.

Common failure modes.

- **Entropy collapse**: entropy drops to 0 suddenly — policy mode-collapsed; increase entropy bonus c_2.
- **Value divergence**: value loss increases unboundedly — learning rate too high or deadly triad.
- **Reward plateau**: reward stops improving — insufficient exploration, local optima, or poor reward design.
- **High KL**: frequent large updates — reduce learning rate or increase clip ε.

Convergence diagnostics. Reliable training requires monitoring several signals:

1. Episode return. Primary metric. Plot rolling mean over last 100 episodes. Look for: monotonic increase, followed by plateau near optimal. Warning signs:

- Flat or decreasing return after initial period: policy collapse; reduce learning rate or check entropy.

- High variance: increase batch size or reduce learning rate.
- Return decreasing sharply: catastrophic forgetting in off-policy methods; check replay buffer.

2. Bellman residual. $\delta = \frac{1}{|\mathcal{B}|}\sum_{(s,a,r,s')\sim\mathcal{B}}(Q(s,a) - y)^2$. Should decrease monotonically during value function learning. Divergence (residual increasing) in off-policy methods indicates the deadly triad: reduce γ, add gradient clipping, or switch to TD3.

3. Policy entropy. For stochastic policies, monitor $\mathcal{H}(\pi(\cdot|s))$. Should stay above target entropy $\bar{\mathcal{H}}$ throughout training (SAC) or decrease gradually (PPO with entropy bonus). Premature collapse to zero entropy indicates mode collapse.

4. KL divergence (PPO). Monitor $D_{\text{KL}}(\pi_{\text{new}} \,\|\, \pi_{\text{old}})$ per update. Should stay near 0.01–0.05. If frequently above 0.1, reduce clipping ε or learning rate. Early stopping when $D_{\text{KL}}(> \,\|\, 1)\ .5\delta_{\text{target}}$ (standard PPO heuristic).

Example — Diagnosing SAC divergence

Symptom: Q-values exploding to $+\infty$ after 10^5 steps. Diagnosis: entropy coefficient α too small (early determinism) + high learning rate = overestimation loop. Fix: increase α target entropy, halve learning rate, add gradient norm clipping at 1.0.

113.7 Reward Shaping Case Study: Sparse Fetch Environments

FetchReach-v2: a 7-DOF robotic arm must reach a target position in 3D space. Sparse reward: $r = 0$ if $\left\|s_{\text{gripper}} - s_{\text{goal}}\right\| < \varepsilon_{\text{success}}$, else $r = -1$. With random exploration: success probability $< 0.01\%$ per episode — essentially zero signal.

With HER (future strategy): Each episode of $T = 50$ steps generates 50 real transitions + $50 \times k = 200$ HER transitions (k=4 additional goals). After 200 episodes: ~ 12000 virtual transitions across diverse goal positions. After 500 episodes: success rate $> 90\%$ on test goals. Without HER: after 10000 episodes, success rate $< 5\%$.

Potential-based shaping for FetchReach. Add shaping $F(s,s') = \gamma\Phi(s') - \Phi(s)$ with $\Phi(s) = -\left\|s_{\text{gripper}} - s_{\text{goal}}\right\|$ (negative distance). This provides dense reward at every step without biasing the optimal policy. With shaping: success rate $> 70\%$ after 100 episodes (without HER). Optimal: combine HER + shaping for fastest learning.

Synthesis: The Mathematical Unity of RL

What we have covered. This volume developed the mathematics of sequential decision-making from first principles to the research frontier. The journey can be summarised as follows.

The Bellman principle is central. Every algorithm in this volume is an instantiation of one idea: the Bellman principle of optimality. Whether we compute value functions by dynamic programming (Parts 89–90), estimate them from samples by TD methods (Parts 91–92), or optimise policies directly via policy gradients (Parts 93–95), we are always exploiting the recursive structure $V(s) = \max_a [r(s,a) + \gamma \mathbb{E}[V(s')]]$. The HJB equation (Part 108) is its continuous-time analogue; the soft Bellman equation (Part 95) is its entropy-regularised variant.

Three fundamental trade-offs.

1. **Exploration vs exploitation**: formalised by the multi-armed bandit (Part 98) and PAC-MDP theory (Part 99). UCB and Thompson sampling resolve it optimally in the bandit setting; in MDPs, count-based bonuses and intrinsic motivation (Part 103) extend these ideas.
2. **Bias vs variance**: TD(λ) and GAE (Parts 91, 94) interpolate between high-bias/low-variance (TD) and low-bias/high-variance (MC) estimation.
3. **Sample efficiency vs computational cost**: model-based methods (Part 96) trade model learning for sample efficiency; model-free methods avoid model error at the cost of more environment interactions.

Connections across Parts.

- MaxEnt RL (Part 95) $\leftrightarrow$ KL-regularised RL (Part 111) $\leftrightarrow$ variational inference $\leftrightarrow$ free energy minimisation (Part 111).
- Offline RL pessimism (Part 107) $\leftrightarrow$ constrained MDP safety (Part 105): both restrict the policy to regions of high confidence.
- Meta-RL (Part 102) $\leftrightarrow$ hierarchical RL (Part 101): both decompose the learning problem across timescales; HRL across action timescales, meta-RL across task timescales.
- IRL (Part 104) $\leftrightarrow$ RLHF (Part 106): both recover reward from behaviour; IRL from expert demonstrations, RLHF from pairwise human preferences.

Open problems. The field remains far from solved: sample efficiency, generalisation across tasks, safe exploration, reward specification, and multi-agent coordination all represent major open challenges. Causal RL (Part 112) and the connections to cognitive science (Part 109) suggest that bridging the gap will require not just better algorithms, but better theories of intelligence itself.

Sparse reward environments. The Fetch robotic environments (Plappert et al., 2018) provide a standard benchmark for sparse reward RL: the agent receives $r = 0$ unless the object reaches the goal within a 5cm threshold, where $r = 1$.

Reward shaping case study. We compare three approaches on FetchPush-v1:

1. **HER only**: relabel trajectories with achieved goal. Converges in $\sim 3 \times 10^6$ steps.
2. **Potential-based shaping**: $\Phi(s) = -d(s_{\text{obj}}, g_{\text{goal}})$. Shaped reward: $\tilde{r} = r + \gamma\Phi(s') -$

$\Phi(s) = r - \gamma d' + d$ (reward progress). Converges in $\sim 1.5 \times 10^6$ steps ($2\times$ speedup). Optimality preserved by theorem.

3. **Non-potential shaping**: $F = +1$ for each novel (s, a) visited. Trains faster but converges to suboptimal policy (maximises exploration reward, not task reward).

Implementation notes.

- Potential-based shaping requires careful normalisation: $\Phi(s)/r_{\text{max}}$ to keep shaped rewards in the same scale as the sparse terminal reward.
- HER + shaping: the HER relabelling must use the *actual achieved goal* for the potential, not the original sparse reward — otherwise the shaped reward is inconsistent with the relabelled goal.
- Threshold sensitivity: the potential function's gradient is steepest near g_{goal}, providing most guidance near the goal — aligned with where the agent needs it most.

Key result. The shaping theorem guarantees that potential-based shaping *cannot* change the optimal policy. In practice, it acts as a curriculum: the smooth gradient signal guides the agent toward the goal region, after which the sparse reward takes over for fine-grained optimisation.

Notation Summary

The following table collects all recurring notation used throughout this volume. Standard set-theoretic and linear-algebraic symbols ($\in, \subset, \cup, \cap, \top, \det, \operatorname{tr}$) are used without special definition. All vectors are column vectors unless transposed.

Core MDP and RL Symbols

Symbol	Type	Meaning
$\mathcal{S}$	set	State space (finite, countable, or $\mathbb{R}^n$)
$\mathcal{A}$	set	Action space
s, s', s_t	$\mathcal{S}$	States (current, next, at time t)
a, a_t	$\mathcal{A}$	Action at time t
$r_t, r(s,a)$	$\mathbb{R}$	Reward at time t / reward function
γ	$(0,1)$	Discount factor
ρ	$\mathbb{R}_{>0}$	Continuous-time discount rate ($\gamma = e^{-\rho \Delta t}$)
$\mathcal{P}(s' \mid s,a)$	$[0,1]$	Transition kernel / probability
$\mathcal{R}(s,a)$	$\mathbb{R}$	Expected reward function
$\pi(a \mid s)$	$[0,1]$	Stochastic policy
$\pi(s)$	$\mathcal{A}$	Deterministic policy
π^*	policy	Optimal (deterministic) policy
G_t, G_t	$\mathbb{R}$	Discounted return $\sum_{k\geq 0} \gamma^k r_{t+k}$
$G_t^{(n)}$	$\mathbb{R}$	n-step return
G_t^{λ}	$\mathbb{R}$	λ-return (geometric mix of n-step)
T, T	$\mathbb{N}$	Episode horizon
τ, τ	sequence	Trajectory $(s_0, a_0, r_0, s_1, ...)$
$V^\pi(s), V^\pi(s)$	$\mathbb{R}$	State-value function under π
$Q^\pi(s,a), Q^\pi(s,a)$	$\mathbb{R}$	Action-value function under π
$V^*(s)$	$\mathbb{R}$	Optimal state-value function
$Q^*(s,a)$	$\mathbb{R}$	Optimal action-value function
$A^\pi(s,a)$	$\mathbb{R}$	Advantage $Q^\pi - V^\pi$
$\hat{A}_t$	$\mathbb{R}$	GAE advantage estimate
$V^\pi_{\text{soft}}, Q^\pi_{\text{soft}}$	$\mathbb{R}$	Soft (entropy-augmented) value functions
$\mathcal{T}$	operator	Bellman optimality operator
$\mathcal{T}^\pi$	operator	Bellman evaluation operator for π
$d^\pi(s)$	$[0,1]$	Discounted state visitation measure
δ_t	$\mathbb{R}$	TD error $r_t + \gamma V(s_{t+1}) - V(s_t)$
$e_t(s)$	$\mathbb{R}_{\geq 0}$	Eligibility trace for state s
λ	$[0,1]$	TD(λ) trace decay; also GAE parameter
$\alpha, \alpha_\theta, \alpha_\phi$	$(0,1)$	Learning rates (general, actor, critic)
α	$\mathbb{R}_{>0}$	Temperature / entropy coefficient (SAC, MaxEnt)
ε	$(0,1)$	Exploration probability (ε-greedy) or PPO clip ratio
τ	$(0,1)$	Soft target update coefficient (DDPG, SAC)
$\square$	$\mathbb{R}^d$	Policy network parameters
$\square$	$\mathbb{R}^d$	Value / critic network parameters
$\square$	$\mathbb{R}^d$	World model parameters
$r_t(\square)$	$\mathbb{R}_{>0}$	PPO probability ratio $\pi_\theta(a_t \mid s_t) / \pi_{\theta_{\text{old}}}(a_t \mid s_t)$
$J(\square), J(\pi)$	$\mathbb{R}$	Performance objective
$\mathcal{H}(\pi(\cdot \mid s))$	$\mathbb{R}$	Policy entropy $-\sum_a \pi \log \pi$
$\mathcal{D}$	set	Replay buffer or offline dataset
$\mathcal{B}$	set	Minibatch sampled from $\mathcal{D}$
$N(s), N(s,a)$	$\mathbb{N}$	Visit count for state / state-action pair
$\mu_i, \hat{\mu}_i$	$\mathbb{R}$	True / empirical mean of bandit arm i
Δ_i	$\mathbb{R}_{\geq 0}$	Suboptimality gap $\mu^* - \mu_i$
$\text{Regret}(T)$	$\mathbb{R}_{\geq 0}$	Cumulative regret over T rounds
$p(\mathcal{M})$	distribution	Task distribution (meta-RL)
$z, \mathbf{z}$	$\mathbb{R}^d$	Latent state (world models, context encoders)
$\Phi(s)$	$\mathbb{R}$	Potential function (reward shaping)
$c(s,a), J_c$	$\mathbb{R}$	Cost function / cumulative cost (safe RL)
ω	$\mathbb{R}^d$	IRL reward parameters

Mathematical Operators and Abbreviations

Symbol	Meaning
$\mathbb{E}_{\pi}[\cdot]$, $\mathbb{E}_{s\sim\mu}[\cdot]$	Expectation under policy π / distribution μ
$\|\cdot\|_{\infty}$, $\|\cdot\|_2$	ℓ^∞ (sup) norm, Euclidean norm
$\|\cdot\|_{\mu}$	Weighted ℓ^2 norm with measure μ
$D_{\mathrm{KL}}(p \,\|\, q)$	Kullback-Leibler divergence $\sum_x p(x)\log(p(x)/q(x))$
$D_{\mathrm{TV}}(p, q)$	Total variation distance $\frac{1}{2}\sum_x \|p(x) - q(x)\|$
∇_θ, ∇	Gradient with respect to θ / gradient operator
$\mathbf{H}_\theta$	Hessian of a scalar w.r.t. θ
$\frac{\partial f}{\partial x}$	Partial derivative of f w.r.t. x
$\triangleq$	Definition (left side defined by right side)
$\arg\max_a f(a)$	Set of maximisers of f
$[n]$	$\{1, 2, \dots, n\}$
$\mathbf{1}[\cdot]$	Indicator function
$\sigma(\cdot)$	Sigmoid function $1/(1+e^{-x})$
$\mathrm{softmax}(\mathbf{x})_i$	$e^{x_i}/\sum_j e^{x_j}$
$\Delta(\mathcal{X})$	Set of probability distributions over $\mathcal{X}$
a.s.	Almost surely (with probability 1)
w.p.	With probability
i.i.d.	Independent and identically distributed
PAC	Probably Approximately Correct
POMDP	Partially Observable MDP
CTDE	Centralised Training, Decentralised Execution
GAE	Generalised Advantage Estimation
KKT	Karush-Kuhn-Tucker (optimality conditions)

Key Theorems Reference

This appendix collects the principal theoretical results proved or stated in this volume, grouped by topic, with cross-references to the main text.

B.1 Dynamic Programming and Optimality

Bellman contraction (§90.1). The Bellman optimality operator $\mathcal{T}$ is a γ-contraction in ℓ^∞: $\|\mathcal{T}u - \mathcal{T}v\|_\infty \le \gamma \|u - v\|_\infty$. By Banach's fixed-point theorem, $\mathcal{T}$ has a unique fixed point V^*, and value iteration $v_{k+1} = \mathcal{T}v_k$ converges geometrically: $\|v_k - V^*\|_\infty \le \gamma^k \|v_0 - V^*\|_\infty$.

Existence of optimal policy (§90.1). For any finite MDP there exists at least one deterministic optimal policy π^* sharing the unique optimal value functions V^* and Q^*. Proof uses compactness of the policy space and the partial order induced by V^*.

Policy improvement theorem (§90.2). If π_{k+1} is the greedy improvement of π_k, then $V^{\pi_{k+1}}(s) \ge V^{\pi_k}(s)$ for all s, with equality iff $\pi_k = \pi^*$. Corollary: policy iteration terminates at π^* in at most $|\mathcal{A}|^{|\mathcal{S}|}$ steps.

Bellman evaluation operator contraction (§89.4). The policy evaluation operator $(\mathcal{T}^\pi v)(s) = \sum_a \pi(a|s)[r(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|s,a)v(s')]$ is a γ-contraction; V^π is its unique fixed point. The matrix solution is $\mathbf{v}^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1}\mathbf{R}^\pi$.

B.2 Temporal Difference and Q-Learning

TD(0) convergence (§91.2). For a fixed policy with step sizes satisfying $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$ (Robbins-Monro), TD(0) converges a.s. to V^π. Proof via stochastic approximation / ODE method (Borkar 2008).

Q-Learning convergence (§91.3). Tabular Q-Learning with Robbins-Monro step sizes and all (s, a) visited infinitely often converges a.s. to Q^* (Watkins & Dayan 1992, Tsitsiklis 1994).

TD fixed point (§92.1). Linear TD with feature matrix $\square$ and on-policy distribution μ converges to $\square^* = \mathbf{A}^{-1}\mathbf{b}$ where $\mathbf{A} = \mathbb{E}[\square(\square - \gamma\square')^\top]$, $\mathbf{b} = \mathbb{E}[\square\, r]$. This is not the MSVE minimum but satisfies $\text{MSVE}(\square^*) \leq \frac{1}{1-\gamma} \min_\square \text{MSVE}(\square)$.

Deadly triad / Baird's example (§99.3). The combination of function approximation + bootstrapping + off-policy training can cause semi-gradient TD to diverge even for linear function approximation. Gradient TD methods (GTD2, TDC) provably converge for all three conditions simultaneously.

B.3 Policy Gradient Methods

Policy gradient theorem (§93.1). For any differentiable policy π_θ and performance objective $J(\theta)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot Q^{\pi_\theta}(s_t, a_t)\right]$$

The gradient is computable without knowing or differentiating through $\mathcal{P}$.

Baseline unbiasedness (§93.2). For any state-dependent baseline $b(s)$: $\mathbb{E}_{a\sim\pi}[\nabla_\theta \log \pi_\theta(a|s) \cdot b(s)] = 0$. Proof uses $\sum_a \pi_\theta(a|s) = 1$.

Performance difference lemma (§94.1). For any two policies $\pi, \tilde{\pi}$: $J(\tilde{\pi}) - J(\pi) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\tilde{\pi}}, a\sim\tilde{\pi}}[A^\pi(s, a)]$.

TRPO monotone improvement (§94.1). If the KL constraint $\mathbb{E}_s[D_{\text{KL}}(\pi(\cdot|s) \,\|\, \tilde{\pi}(\cdot|s))] \leq \delta$ is satisfied, then $J(\tilde{\pi}) \geq L^\pi(\tilde{\pi}) - C\delta$ where $C = 4r_{\max}\gamma/(1-\gamma)^2$, guaranteeing monotone improvement.

Deterministic policy gradient (§95.3). For deterministic μ_θ: $\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}[\nabla_\theta \mu_\theta(s) \cdot \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}]$.

B.4 Bandits and Exploration

Lai-Robbins lower bound (§98.1). For any consistent bandit algorithm: $\liminf_{T\to\infty} \text{Regret}(T)/\log T \geq \sum_{i:\Delta_i>0} \Delta_i / d(\mu_i, \mu^*)$, where d is the KL divergence. This is an information-theoretic lower bound on exploration.

UCB1 regret bound (§98.2). UCB1 achieves $\mathbb{E}[\text{Regret}(T)] \leq \sum_{i:\Delta_i>0}[8\ln T/\Delta_i + (1+\pi^2/3)\Delta_i]$, which is $O(\sqrt{KT\ln T})$ when all gaps are equal. UCB1 is asymptotically optimal.

Thompson Sampling optimality (§98.3). Thompson Sampling achieves the Lai-Robbins lower bound asymptotically for exponential family reward distributions (Korda et al. 2013).

B.5 Sample Complexity and Learning Theory

PAC-MDP definition (§99.1). An algorithm is PAC-MDP if it takes at most $\text{poly}(|\mathcal{S}|, |\mathcal{A}|, 1/\varepsilon, 1/\delta, 1/(1-\gamma))$ non-ε-optimal steps with probability $\geq 1 - \delta$.

RMAX sample complexity (§99.1). RMAX is PAC-MDP with sample complexity $\tilde{O}(|\mathcal{S}|^2|\mathcal{A}|/[\varepsilon^3(1-\gamma)^6])$.

UCBVI regret (§99.2). In the episodic setting, UCBVI achieves $\text{Regret}(K) = \tilde{O}(\sqrt{H^2|\mathcal{S}||\mathcal{A}|K})$, near-matching the $\Omega(\sqrt{H|\mathcal{S}||\mathcal{A}|K})$ lower bound.

Potential-based shaping invariance (§103.1). For any potential $\Phi : \mathcal{S} \to \mathbb{R}$, the shaped reward $\tilde{r} = r + \gamma\Phi(s') - \Phi(s)$ preserves the set of optimal policies exactly.

B.6 Advanced Topics

CMDP Lagrangian duality (§105.1). The constrained MDP $\max_\pi J_r(\pi)$ s.t. $J_{c_k}(\pi) \le d_k$ is equivalent under strong duality to $\min_{\lambda \ge 0} \max_\pi [J_r(\pi) - \sum_k \lambda_k (J_{c_k}(\pi) - d_k)]$. Duality gap is zero when the policy class is convex (linear policies in finite MDPs).

MaxEnt optimal policy (§95.1). Under the maximum entropy framework, the optimal soft policy is $\pi^*(a|s) = \exp(Q^*_{\text{soft}}(s,a)/\alpha)/Z(s)$, a Boltzmann distribution over action values.

IRL identifiability (§104.1). The reward function is not identifiable from behaviour alone: any reward in the equivalence class $\{r' : V^{\pi^*}_{r'}(s) \ge V^{\pi}_{r'}(s)\ \forall \pi, s\}$ rationalises the expert's policy. Maximum entropy IRL (Ziebart et al. 2008) selects the reward maximising policy entropy, providing a canonical choice.

QMIX IGM condition (§97.3). QMIX satisfies the Individual-Global-Max property: $\arg\max_{\mathbf{a}} Q_{\text{tot}} = (\arg\max_{a_1} Q_1, \dots, \arg\max_{a_n} Q_n)$, enabling decentralised greedy execution while training a centralised joint value function.

Mathematical Background

This appendix collects the mathematical prerequisites used repeatedly in the volume.

C.1 Banach Fixed-Point Theorem

Banach Fixed-Point Theorem

Let (X, d) be a complete metric space and $T : X \to X$ a contraction with constant $c \in [0, 1)$: $d(Tx, Ty) \le c \cdot d(x, y)$ for all $x, y \in X$. Then T has a unique fixed point $x^* = Tx^*$, and the iteration $x_{k+1} = Tx_k$ converges to x^* for any $x_0 \in X$, with rate $d(x_k, x^*) \le c^k d(x_0, x^*)/(1-c)$.

Application: the Bellman operators $\mathcal{T}$ and $\mathcal{T}^\pi$ are contractions with constant γ in $(C(\mathcal{S}), \|\cdot\|_\infty)$, which is complete. This immediately gives existence, uniqueness, and geometric convergence of value iteration and policy evaluation.

C.2 Stochastic Approximation

Robbins-Monro theorem. Consider the recursion $x_{k+1} = x_k + \alpha_k(f(x_k) + \xi_k)$ where f has a root x^*, ξ_k is zero-mean noise, and step sizes satisfy:

$$\sum_{k=1}^{\infty} \alpha_k = \infty, \qquad \sum_{k=1}^{\infty} \alpha_k^2 < \infty$$

Under regularity conditions, $x_k \to x^*$ a.s. TD(0) and Q-Learning are instances: f is the Bellman residual, ξ_k is sampling noise. Typical step sequences: $\alpha_k = 1/k$ (satisfies conditions); $\alpha_k = 0.01$ constant (violates $\sum \alpha_k^2 < \infty$; converges in mean to a neighbourhood of x^*).

ODE method (Borkar 2008). The continuous-time limit of the stochastic approximation $x_{k+1} = x_k + \alpha_k h(x_k, \xi_k)$ is the ODE $\dot{x} = \bar{h}(x)$ where $\bar{h}(x) = \mathbb{E}_\xi[h(x, \xi)]$. If x^* is a globally asymptotically stable equilibrium of the ODE, then $x_k \to x^*$ a.s.

C.3 Measure Theory and Probability

Law of total expectation. $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|Y]]$. Applied in the Bellman equation derivation: $\mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi[r_t + \gamma G_{t+1}|s_t = s] = \sum_a \pi(a|s)\mathbb{E}[r_t + \gamma V^\pi(s_{t+1})|s_t = s, a_t = a]$.

Hoeffding's inequality. For i.i.d. bounded random variables $X_i \in [a_i, b_i]$:

$$P\left(\frac{1}{n}\sum_{i=1}^{n} X_i - \mathbb{E}[X_i] \geq t\right) \leq \exp\left(\frac{-2n^2t^2}{\sum_i (b_i - a_i)^2}\right)$$

Used in UCB analysis: with $t = \sqrt{2 \ln T / n_i}$ and range $[0, 1]$, the bound on overestimation is $1/T^2$.

Log-derivative trick (REINFORCE). For a parametric distribution $p_\theta(x)$: $\nabla_\theta p_\theta(x) = p_\theta(x) \nabla_\theta \log p_\theta(x)$. This converts gradient of expectation to expectation of score function: $\nabla_\theta \mathbb{E}_{p_\theta}[f(x)] = \mathbb{E}_{p_\theta}[f(x) \nabla_\theta \log p_\theta(x)]$.

C.4 Convex Optimisation and Duality

KKT conditions. For the constrained problem $\min_x f(x)$ s.t. $g_i(x) \leq 0$ and $h_j(x) = 0$, a point x^* is optimal if there exist multipliers $\lambda_i \geq 0$, ν_j such that: (i) stationarity: $\nabla f(x^*) + \sum_i \lambda_i \nabla g_i(x^*) + \sum_j \nu_j \nabla h_j(x^*) = 0$; (ii) primal feasibility: $g_i(x^*) \leq 0$, $h_j(x^*) = 0$; (iii) dual feasibility: $\lambda_i \geq 0$; (iv) complementary slackness: $\lambda_i g_i(x^*) = 0$. Applied in TRPO (KL-constrained policy optimisation) and CMDP (cost-constrained RL).

Conjugate duality. The Lagrangian dual of $\min_x f(x)$ s.t. $g(x) \leq 0$ is $\max_{\lambda \geq 0} \min_x [f(x) + \lambda g(x)]$. Strong duality (primal = dual) holds under Slater's condition: existence of a strictly feasible x with $g(x) < 0$. Used in SAC temperature tuning and CMDP Lagrangian formulations.

C.5 Information Theory

KL divergence. $D_{\mathrm{KL}}(p \,\|\, q) = \mathbb{E}_p[\log(p/q)] \geq 0$ with equality iff $p = q$ a.e. (Gibbs inequality). Not symmetric: $D_{\mathrm{KL}}(p \,\|\, q) \neq D_{\mathrm{KL}}(q \,\|\, p)$ in general.

Shannon entropy. $H(p) = -\sum_x p(x) \log p(x) = \log |\mathcal{X}| - D_{\mathrm{KL}}(p \,\|\, \text{Uniform})$. The maximum entropy distribution over a finite set is uniform; over $\mathbb{R}$ with variance constraint it is Gaussian.

Bradley-Terry model. For pairwise preferences, $P(y_1 \succ y_2|r) = \sigma(r(y_1) - r(y_2))$ where σ is the sigmoid. Used in RLHF reward model training to convert human preferences into a scalar reward. The log-likelihood is $\ell(r) = \sum_{(y_w, y_l)} \log \sigma(r(y_w) - r(y_l))$, a concave function in r.

Fisher information matrix. $\mathbf{F}(\theta) = \mathbb{E}_{x \sim p_\theta}[\nabla_\theta \log p_\theta(x) (\nabla_\theta \log p_\theta(x))^\top] = -\mathbb{E}[\mathbf{H}_\theta \log p_\theta(x)]$. The natural gradient $\mathbf{F}^{-1} \nabla J$ is the steepest ascent direction in the KL geometry, used in Natural Policy Gradient and TRPO.

References

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2), 235–256.

[2] M. G. Azar, I. Osband, and R. Munos (2017). Minimax regret bounds for reinforcement learning. *ICML 2017*.

[3] L. Baird (1995). Residual algorithms: Reinforcement learning with function approximation. *ICML 1995*.

[4] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos (2016). Unifying count-based exploration and intrinsic motivation. *NeurIPS 2016*.

[5] R. Bellman (1957). *Dynamic Programming*. Princeton University Press.

[6] R. I. Brafman and M. Tennenholtz (2002). R-MAX — A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3, 213–231.

[7] D. Hafner, J. Lillicrap, M. Norouzi, and J. Ba (2023). Mastering diverse domains through world models. *arXiv:2301.04104*.

[8] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine (2018). Soft Actor-Critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *ICML 2018*.

[9] S. Kakade and J. Langford (2002). Approximately optimal approximate reinforcement learning. *ICML 2002*.

[10] S. Levine, A. Kumar, G. Tucker, and J. Fu (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv:2005.01643*.

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2016). Continuous control with deep reinforcement learning. *ICLR 2016*.

[12] H. R. Maei, C. Szepesvári, S. Bhatnagar, D. Precup, D. Silver, and R. S. Sutton (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. *NeurIPS 2009*.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529–533.

[14] T. Rashid, M. Samvelyan, C. Schroeder de Witt, G. Farquhar, J. Foerster, and S. Whiteson (2018). QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. *ICML 2018*.

[15] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz (2015). Trust region policy optimization. *ICML 2015*.

[16] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel (2016). High-dimensional continuous control using generalized advantage estimation. *ICLR 2016*.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). Proximal policy optimization algorithms. *arXiv:1707.06347*.

[18] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller (2014). Deterministic policy gradient algorithms. *ICML 2014*.

[19] A. L. Strehl, L. Li, and M. L. Littman (2009). Reinforcement learning in finite MDPs:

PAC analysis. *Journal of Machine Learning Research*, 10, 2413–2444.

[20] R. S. Sutton (1988). Learning to predict by the method of temporal differences. *Machine Learning*, 3(1), 9–44.

[21] R. S. Sutton (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4), 160–163.

[22] R. S. Sutton and A. G. Barto (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.

[23] W. R. Thompson (1933). On the likelihood that one unknown probability exceeds another. *Biometrika*, 25(3–4), 285–294.

[24] J. N. Tsitsiklis and B. Van Roy (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5), 674–690.

[25] H. van Hasselt, A. Guez, and D. Silver (2016). Deep reinforcement learning with double Q-learning. *AAAI 2016*.

[26] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas (2016). Dueling network architectures for deep reinforcement learning. *ICML 2016*.

[27] C. J. C. H. Watkins (1989). *Learning from Delayed Rewards* (PhD thesis). University of Cambridge.

[28] R. J. Williams (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4), 229–256.

[29] J. Achiam, D. Held, A. Tamar, and P. Abbeel (2017). Constrained policy optimization. *ICML 2017*.

[30] Y. Bai, S. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, N. Joseph, S. Kadavath, J. Kernion, T. Conerly, S. El-Showk, N. Elhage, Z. Hatfield-Dodds, D. Hernandez, T. Hume, S. Johnston, S. Kravec, L. Lovitt, N. Nanda, C. Olsson, D. Amodei, T. Brown, J. Clark, S. McCandlish, C. Olah, B. Mann, and J. Kaplan (2022). Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv:2204.05862*.

[31] C. Finn, P. Abbeel, and S. Levine (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *ICML 2017*.

[32] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine (2017). Reinforcement learning with deep energy-based policies. *ICML 2017*.

[33] M. Janner, J. Fu, M. Zhang, and S. Levine (2019). When to trust your model: Model-based policy optimization. *NeurIPS 2019*.

[34] H. Lee, S. Phatale, H. Mansoor, K. Lu, T. Mesnard, C. Bishop, V. Carbune, and A. Rastogi (2023). RLAIF: Scaling reinforcement learning from human feedback with AI feedback. *arXiv:2309.00267*.

[35] A. Y. Ng, D. Harada, and S. Russell (1999). Policy invariance under reward transformations: Theory and application to reward shaping. *ICML 1999*.

[36] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell (2017). Curiosity-driven exploration by self-supervised prediction. *ICML 2017*.

[37] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn (2023). Direct preference optimization: Your language model is secretly a reward model. *NeurIPS 2023*.

[38] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey (2008). Maximum entropy inverse reinforcement learning. *AAAI 2008*.

[39] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving (2019). Fine-tuning language models from human preferences. *arXiv:1909.08593*.